A Statistical Analysis of a Midi File Database

Seymour Shlien

December 18, 2018

1 Introduction

MIDI is a standard for communicating and representing musical events to a hardware or software device [1], [2]. The midi file contains a list of note-on and note-off and numerous controller commands that allow a device to synthesize the music encoded. These commands specify pitch, loudness and duration of each note. Frequently the notes are quantized into musical intervals, permitting the midi file to be converted into a symbolic representation such as sheet music. Each of the notes refers to a specific MIDI channel which is mapped to a particular musical instrument. The MIDI standard allows for 16 MIDI channels, and specifies 128 musical instruments.

There are large collections of midi files that are freely available on the internet. For example, Colin Raffel has placed a large collection of 176,581 midi files [3] gathered from numerous sources. Many of these files were gathered for the purpose of matching them to the Million Song Dataset[4] and [5]. This paper deals with a subset of of this collection, called 'Clean MIDI subset'. The subset contains more than 17,200 midi files organized by the artist. The collection was used to study the application of various statistical and data exploration techniques.

This subset contains a wide selection of popular music grouped among 2197 artists. These artists include well known groups such ABBA, The Beatles, The Rolling Stones as well as many lesser known. The music genres range from different varieties of rock, pop, new wave, hip hop, punk, rap, jazz and classical. The music titles are well known and a lot of supplementary information is available from other sources such as Wikipedia.

Popular music can be classified into hundreds of music genres [6] and [7]. Though there has been considerable research in developing automatic methods for classifying music, for example [8] and [9], their accuracies are limited. Many of the terms used to describe music such as 'Alternative rock' cover diverse styles such as punk rock and new wave. Artist frequently fuse various styles.

Though most of the work on digital music collections deal with audio representations, there are are few studies dealing with symbolic representations such as MIDI files. Several packages such as jMIR [8], Humdrum [10], and music21 [11] for analyzing this data are available.

In order to explore this large collection, I created a computer program called Midiexplorer that is available on SourceForge.net [12]. The program was developed to browse the collection, extract various characteristics of the individual files, and search for files that match certain criteria. The contents of a midi file can be viewed in various graphical forms. The program can export various midi file descriptors for further statistical analysis using other packages. A description of this software can be found on the internet [13].

The paper leans towards the application of various statistical analysis techniques for the analysis of such a collection of data. All of these methods are available in the R statistical packages [14] and require only Many of these programs only a few seconds to process this data on a desktop computer. This is an ideal environment for exploring this data and becoming familiar with the many analysis techniques. Since this paper covers many different analysis schemes, it is not practical to give a detailed description of these techniques; however, numerous references are provided. Furthermore, the reader can find many tutorial videos on YouTube.

In many cases, the goal of the analysis was to to separate the collection into distinct groups using the R software. Unfortunately, is not easy to determine whether these groups correspond to musical classes.

The next section describes a method for grouping the midi files on the basis of the musical instruments (MIDI programs) assigned to the different tracks. This is followed by the next section where we attempt to group the midi files on the basis of their pitch class distributions. In the last section some of the characteristics of the percussion tracks are investigated.

2 MIDI Programs

The General MIDI Standard defines 128 programs or musical instruments. These instruments provide the tone colour to the music and give it a certain style. For example, the piano, bass and saxaphone are frequently associated with jazz music. Similarly, the overdriven and distortion guitar are associated with heavy metal. Such tendencies have been exploited for identifying the music genre [15].

Other important features such as tempo, rhythm, and percussion also important in characterizing the music style. Here, we will focus exclusively on the MIDI programs in order to establish their value.

It is not sufficient to determine the absence or presence of these programs in the MIDI file. For example, the flute may occur in just the introduction of the piece. For this reason, the amount of activity in each of the 128 MIDI programs was determined and represented by a program vector.

To compute the activity for a program, we summed the duration of all the notes associated with that program and then divided this sum by the total duration of the midi file. Thus if a single note plays for half the duration of the file, it contributes an activity of 0.5. Chords containing two notes contribute double the amount. The variability of the program vectors were reduced by normalizing them to unit length. This implies that how the activity is distributed among the 128 programs is considered more important rather than the total activity.

More specifically, assume the midi file consists of N notes whose durations are $dur(i)$ and whose program assignments are prg(i). Let $\delta(j,k)$ be the delta function which is 1 when $j = k$ and 0 otherwise. Then the activity for class k is given by

$$activity(k) = \frac{1}{D}\sum_{i=1}^{N} dur(i) * \delta(prg(i), k) \tag{1}$$

where D is the duration of the entire midi file. The program vector consists of the 128 activity values which were normalized so that the vector has unit length. Program vectors were computed for each of the midi files and stored in a database.

The set of all of these vectors can be viewed as a cloud of points in a high dimensional space; however, we can gather their characteristics from statistical analysis. Segmenting this data into various groups having common properties makes it easier to deal with this information.

It is likely that the program vectors can fit into a lower dimensional subspace. For example, many programs such as 'Gun Shot' rarely occur and other programs tend to occur together. The principal components is a linear transform that decorrelates these components and concentrates the information into a lower dimensional space [16]. The principal components is an orthonormal set of vectors that maximize the variances of the data. The R function prcomp() was used to determine this transformation.

As seen in Figure 1, most of the variation is concentrated in the first few components. The MIDI programs contributing the most information were mainly the keyboard, electric guitar, the acoustic guitar, and the string ensemble – Figure **??**.

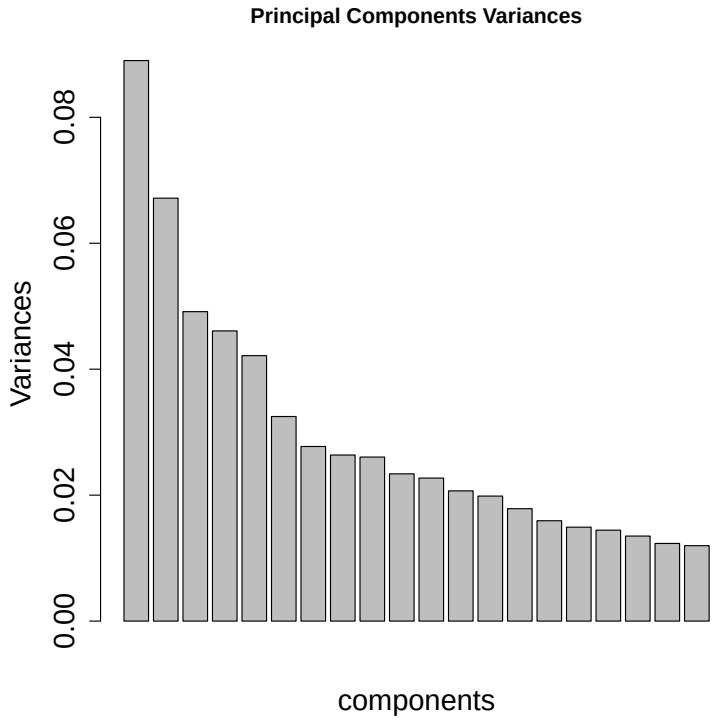

Figure 1: Variances of the principal components of program vectors.

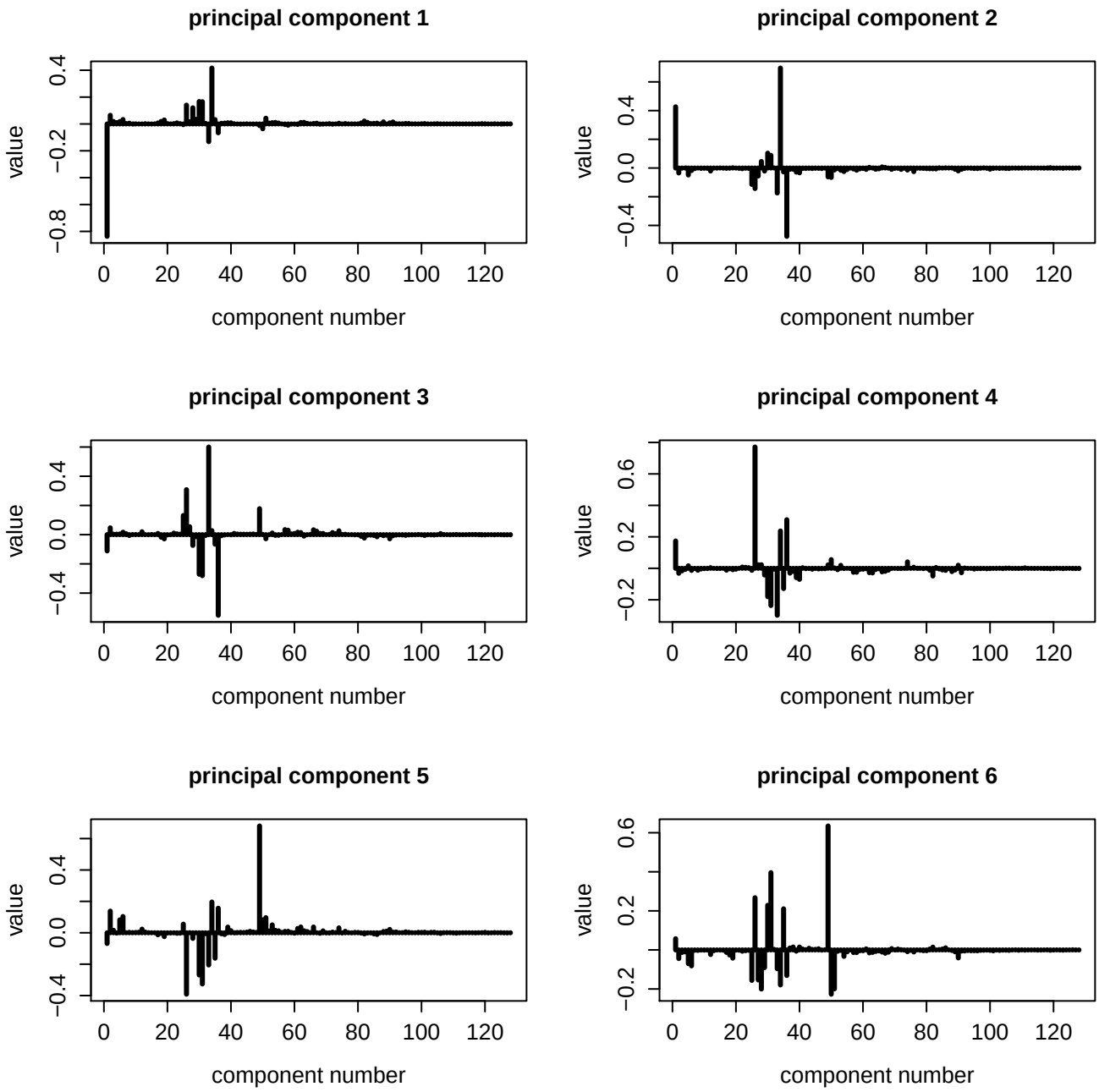

Figure 2: First six principal vectors of program vectors. The horizontal axis indicates the MIDI program number and the vertical scale is the weighting factor of the MIDI program for the particular principal component. The Acoustic Grand, Acoustic Guitar, Overdriven Guitar, Acoustic Bass, Electric Bass (finger), and Fretless Bass MIDI programs are weighted the most in the main principal components.

One can get a glimpse of how the 17200 program vectors are distributed by projecting them onto the principal components. The program vectors were transformed into the coordinates of the first few principal components and plotted in Figure 3.

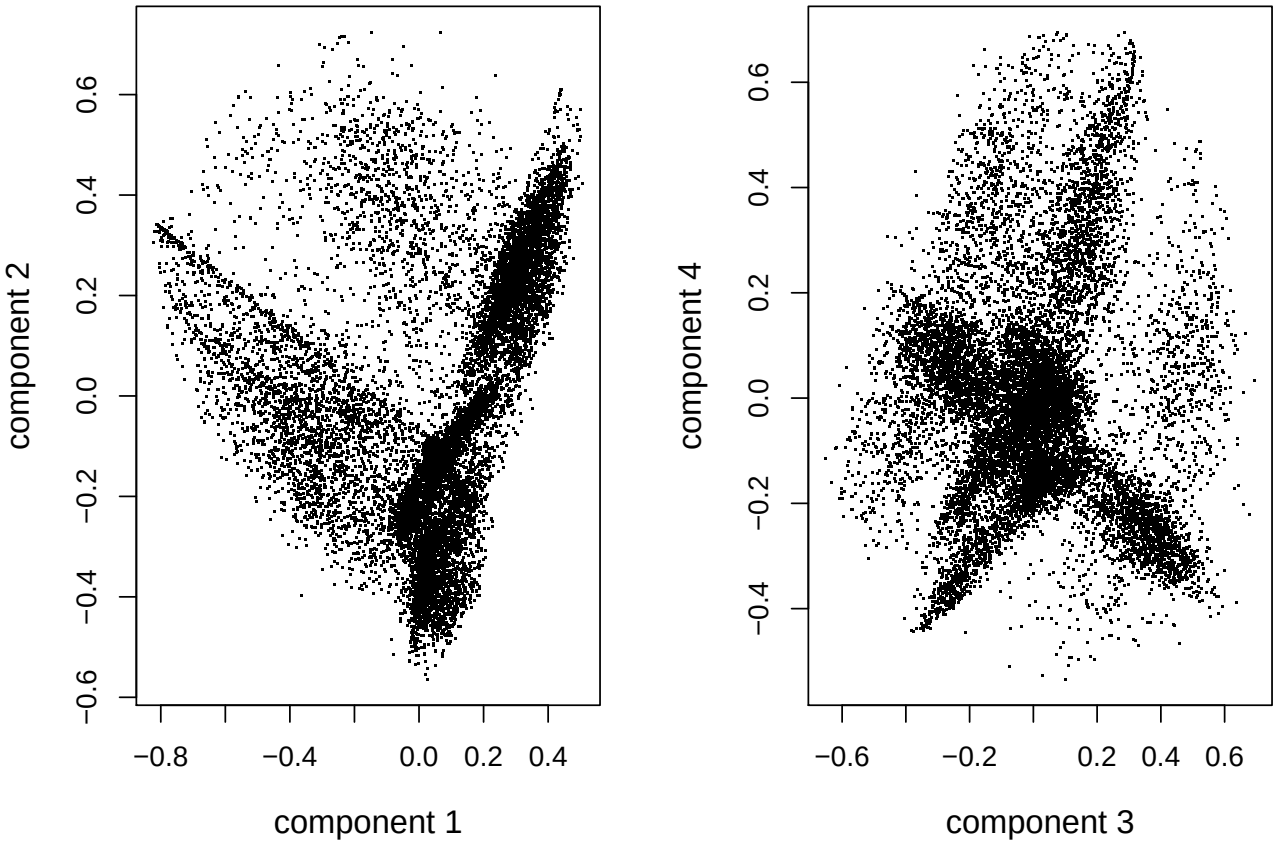

Figure 3: Program vectors projected onto the first 4 principal component.

It is apparent that there are regions of high and low concentration. Though the clusters do not appear distinct, they may be separated in higher dimensions. DBSCAN (Density-based spatial cluster of applications with noise) [17] and [18] is a clustering algorithm that finds regions of high density surrounded by low density regions.

The dbscan function requires the user to specify two parameters – the size of a neighbourhood *eps*, and the minimum size of a cluster *minPts*. (Methods on choosing these parameters are suggested in the documentation.) Given *eps*, these algorithms count the the number of nearest neighbours around each data vector. If the number of neighbours exceeds *minPts*, the data vector is classified as a core point. A point p is directly density-reachable from a core point q if p is in the *eps* neighbourhood of q. A point p is density-reachable from point q with respect to *eps* and *minPts* if there is a chain of points directly reachable from p [19]. All the remaining points are considered as noise points. Dbscan labels all the given data points and identifying clusters.

The OPTICS [20] algorithm, an extension of dbscan, was used in this study. It can reveal cluster structure in regions where are local variations in the densities. Instead of specifying the *eps* parameter, the user specifies the maximum expected *eps* for computational efficiency.

Figure 4 shows all the core points and points reachable from the core points. Due to the sparsity of the data in the high dimensionable space more than 90 percent of the program vectors were classed as noise points. The selected points appear to match the samples in high density regions seen in Figure 3. The function extractDBSCAN() grouped the results into 20 or so clusters. 8 of these clusters were color coded and shown in Figure 5.

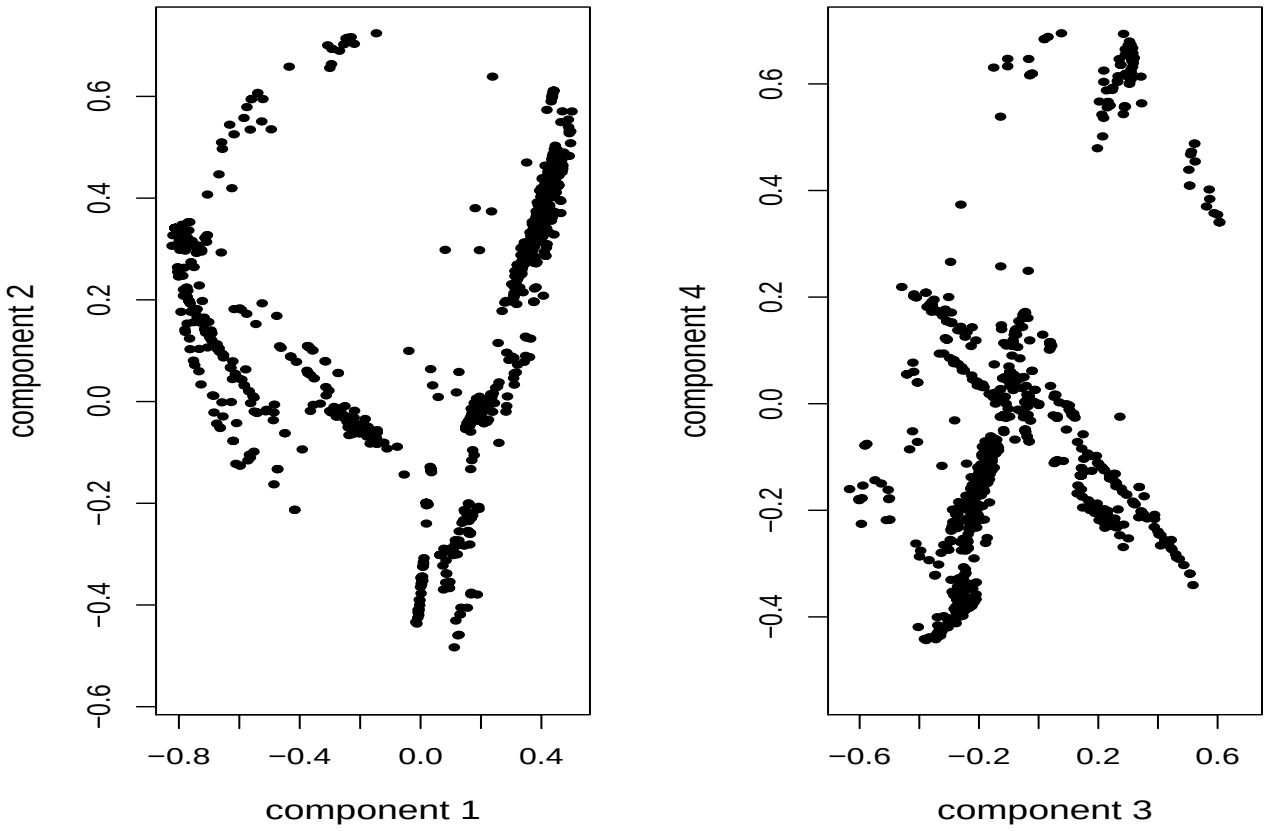

Figure 4: Clustered data returned by OPTICS and extractDBSCAN in the dbscan r package with an $eps = 0.75, minPts = 12$ and $eps_cl = 0.30$.

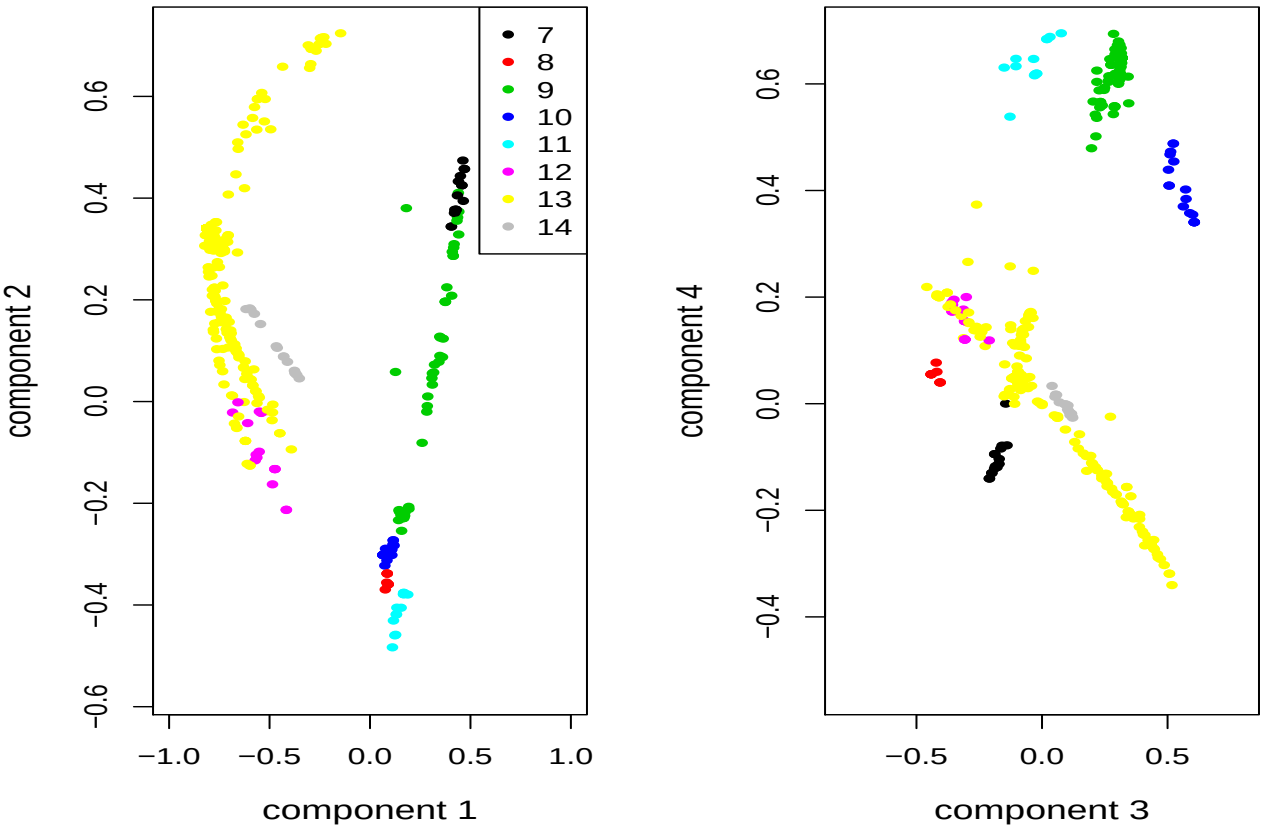

Figure 5: Clustered results for 8 of the clusters (distinguished by color) returned by OPTICS and extractDBSCAN in the dbscan r package with an $eps = 0.75, minPts = 12$ and $eps_cl = 0.30$.

The OPTICS/extractDBSCAN algorithms failed to classify 90 percent of the data points. As the dimensionality of the data increases, the interspacing between the data points tends to become large but more or less the same dimensionality [21] and [22]. The paper [23] is a particular lucid description of how this effects clustering algorithms. As a result of this tendency, it becomes difficult to chose an *eps* value that is large enough to group most of the data samples but not too large too clump all the samples into one cluster.

Though the OPTICS/extractDBSCAN results show the general structure of the program vector distribution, these results were not particular useful. The points in the clusters mapping to the individual MIDI files did not reveal any particular pattern. The samples in some of the clusters (for example, cluster 13 in Figure 5) covered a large region in program space preventing the identification of any coherent pattern. Furthermore, it was preferable to be able to group all of the MIDI files rather than a select few. Therefore, it was decided to try a partitioning cluster scheme.

The k-means algorithm introduced by Lloyd in 1957 [24] is the most well-known method for partitioning data. Like many other clustering techniques, it partitions the data into groups that do not necessarily reflect the regions of high concentration in the distribution. The algorithm tries to minimize the withen cluster dispersion using an iterative algorithm.

The k-means returns cluster centroids that are the average of the cluster sample positions in the data space. In some cases, these centroids may occur in sparse regions far from any particular data sample. It would be advantageous if the cluster centers were actual data samples. Fortunately, the k-medoids algorithm introduced by Kaufman and Rousseeuw [25] does this exactly. It has been demonstrated that the k-medoids algorithm is less sensitive to noisy data and outliers [26]. The k-medoids algorithm is implemented by the pam() function in the R

'Cluster Package'.

There are two remaining issues. First, most cluster partitioning algorithms including the k-medoid require the user to specify the number of clusters. Secondly, the computation time to cluster the 17,200 midi files was excessive.

The determination of the number clusters in a data set is a big topic and there are numerous algorithms and publications on this subject [27], [28] and [29]. Many of the papers compare the accuracy of these algorithms using computer generated data [30] and [31].

There are many tutorials on this topic. The STHDA site [32] supported by A. Kassambara, posts numerous introductory articles on cluster analysis. The class notes for the Stat133 course at Berkeley University of California posted by Phil Spector is also noteable [33]. In addition, there is also the set of slides for the data mining summer course given at University of Michigan by Ala Al-Fuqaha [34]. There are also numerous YouTube videos. The R 'cluster package' comes with 70 pages of documentation [35] containing a lot of useful information.

I had attempted to apply several methods for estimating the number of clusters in the data. Most of the methods validated less than a handful of clusters which was not adequate for this study. At the very end, I decided to partition the data into 30 groups – mainly because it was barely manageable for this study.

The computation time for the k-medoids algorithm was a second issue. Unfortunately, the k-medoids algorithm requires large memory resources resources and runs fairly slow. The processing time is proportional to $k(N-k)^2$ where k is the number of clusters to be extracted and N is the number of data vectors.

Kaufman and Rousseeuw also introduced an alternative approach called clara (Clustering for Large Applications) It can cope with large amounts of data [25] and runs very fast on a desktop computer. The scheme reapplies the k-medoids algorithm on numerous subsets of the data and returns the best results. Unlike the k-medoid algorithm, it does not find the optimum configuration; however, the results were adequate for this study.

The data was partitioned into 30 groups using the clara algorithm in the 'cluster package'.

In addition to mapping the data vectors to the cluster numbers, both the pam (k-medoids) and the clara algorithm return various measures of the cluster quality which are listed in Table 1. *size* indicates the number of data samples that were assigned to the cluster. *max_dis* specifies the maximum distance of the data samples assigned to the specific cluster to its medoid. *avg_dis* is the average distance of the data samples assigned to the specific cluster to its medoid. *ratio* is the ratio of *max_dis* over the minimum distance of all the data samples not assigned to that specific cluster to the cluster medoid. If *ratio* is small the cluster is well-separated from the other clusters. The most common MIDI programs associated with the cluster is also given.

cluster	programs	size	max_dis	avg_dis	ratio
1	Dst Guitar, Elec Bass,	783	1.21	0.84	1.86
2	Act Guitar, Act Bass, ...	467	1.28	0.88	1.48
3	String Ens, Wind ...	917	1.39	0.85	1.42
4	Elec Guitar, Elec Bass, ...	672	1.37	0.85	1.41
5	Act Grand	1537	1.37	0.43	1.58
6	Act Guitar, Elec Guitar, ...	853	1.24	0.82	1.39
7	Synth Bass ...	382	1.36	1.01	1.43
8	Synth Bass, Lead, Choir, ...	354	1.40	1.07	1.09
9	Brass and Wind,	75	1.36	1.10	1.14
10	Elec Bass, Elec Guitar, Pad ...	333	1.26	0.88	1.12
11	String Ens, ...	729	1.41	0.86	1.53
12	Ovr Guitar, Elec Bass ...	996	1.34	0.79	1.46
13	Dst Guitar, Elec Bass, ...	574	1.37	0.77	2.27
14	Fretless Bass, ...	1109	1.41	0.92	1.79
15	Act Grand and Act Bass,	1082	1.37	0.82	1.49
16	Act Grand, Act Bass, Sax, ...	238	1.38	0.76	1.18
17	Dst Guitar, Elec Bass, Choir, ...	275	1.18	0.58	1.96
18	Elec Piano, Elec Bass, ...	400	1.41	0.92	1.43
19	Act Guitar	862	1.40	0.77	1.43
20	Fretless Bass, String Ens, ...	630	1.41	0.81	1.63
21	Elec Guitar, Bright Act, ...	360	1.37	0.86	1.42
22	Act Guitar	355	1.31	0.81	1.41
23	Act Guitar, Elec Guitar, ...	417	1.27	0.89	1.47
24	Dst Guitar, Fretless Bass, ...	298	1.39	0.79	1.76
25	Elec Piano, Fretless Bass, ...	550	1.40	0.91	1.51
26	Bright Act, Electric Bass, ...	398	1.32	0.88	1.36
27	Act Grand, Fretless Bass, ...	755	1.38	0.86	1.35
28	Elec Grand, Elec Bass ...	236	1.22	0.82	1.10
29	Elec Guitar, Dst Guitar, Sax, ...	329	1.33	0.84	1.80
30	Clav, Dst Guitar, Voice, ...	133	1.38	1.11	1.15

Table 1: 30 program clusters, their significant programs, number of samples assigned to the clusters, and other cluster descriptors. The following abbreviations are used here: Elec - Electric, Act - Acoustic, Ens - Ensemble, Dst - Distortion, Ovr - Overdriven.

Clusters varied in size from a 1000 to less than a 100. The data in the large clusters were generally similar, while the smaller clusters were more mixed and spread out over a larger region in the program activity space. The cluster *ratio* values all exceeded 1.0, indicating that the clusters were not well separated.

The medoids of the 30 cluster model were transformed to the principal components and plotted in Figure 6 so that we could visualize the positions of the medoids.

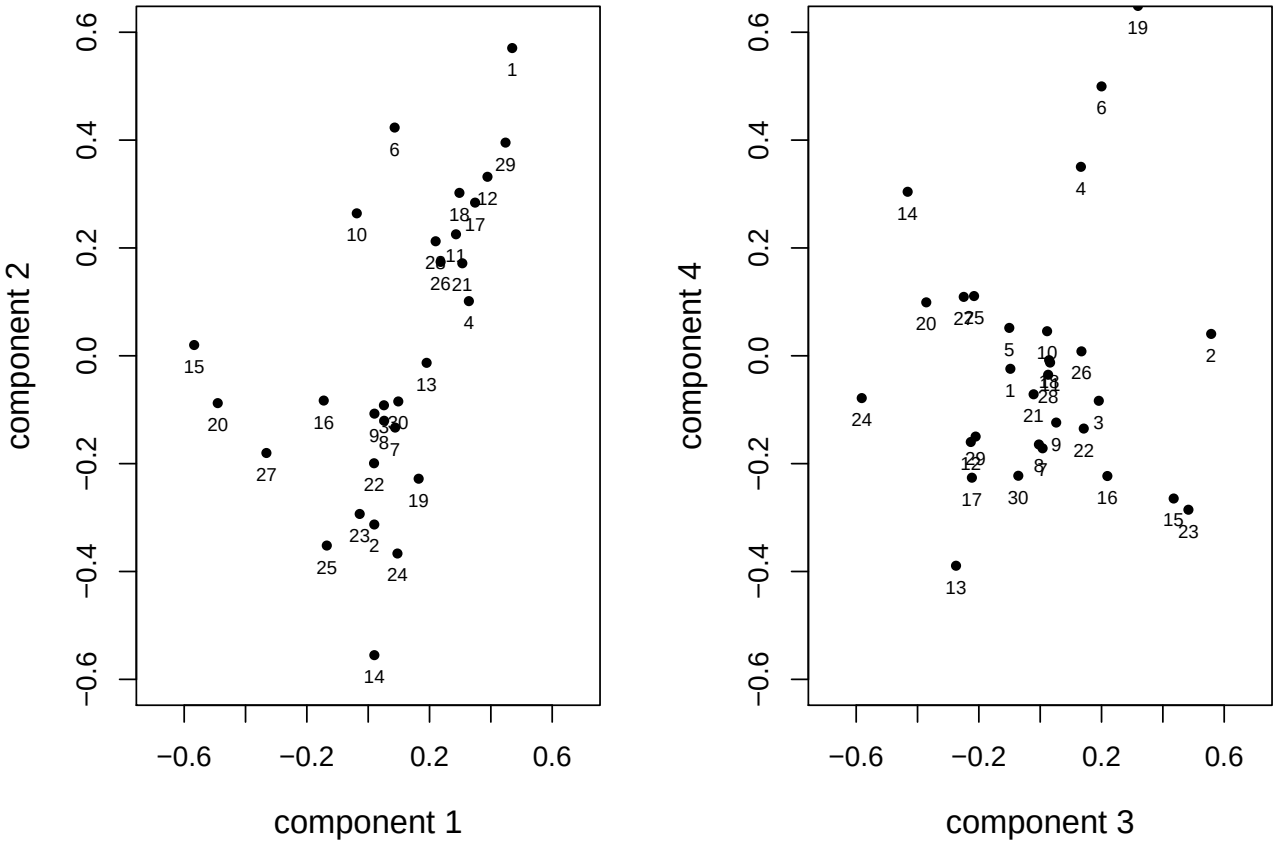

Figure 6: Projection of the program clusters onto the first 4 principal components.

The clusters tend to separate the midi files into music with similar styles. For example, Cluster 1 with the Distortion Guitar and Electric Bass has hard rock, grunge, heavy metal type of music. Cluster 3 which has String Ensembles and possibly wind or brass instruments tend to attract music from the baroque and classical era. Cluster 4 which is heavy on the guitars seems associated with folk and country music. Table 2 lists the midi files centered around program clusters 15 and 16. The midi file which heads the group is the medoid of that cluster. The r value measures the discrepancy of the other midi files in the group. r Values close to zero are good matches. The particular assemblage of musical instruments in these clusters lead to many jazz compositions. Genre assignment were obtained from Wikipedia by searching the title and artist.

Table 3 illustrates a different style.

3 Tonality Analysis

Jazz and Blues music have their own particular scales [36]. The nature of the music is determined by whether it is played in a major, minor, pentatonic, chromatic scale. Counting all the naturals flats and sharps, the Western music scale consists of 11 pitch classes. Most music uses a subset of eight of these pitch classes forming an octave and the particular selection determines whether the music is in a major, minor or other scale. Western music in particular popular music has a tonal center which defines the key of the piece.

The pitch class vector was determined by measuring the activity in each of the pitch classes. Here activity is measured by the amount of time the pitch is played relative to the total length of the midi file. This was done in

r	file	genre
15	Oscar Peterson/Emily.1.mid	Jazz
	Acoustic Grand and Acoustic Bass	1082
0.000	Bee Gees/Israel.mid	Pop
0.000	Elvis Presley/Heartbreak Hotel.mid	Blues
0.000	Porter Cole/I Concentrate on You.mid	
0.000	Porter Cole/You'd Be So Nice to Come Home To.mid	Jazz
0.000	Tyner Mccoy/You'd Be So Nice to Come Home To.mid	Jazz
0.002	Oscar Peterson/I Want to Be Happy.mid	Jazz
0.003	Evans Bill/Israel.mid	Jazz
0.004	Gershwin/A Foggy Day.mid	Jazz
0.005	Queen/My Melancholy Blues.mid	Progressive rock
0.008	Powell Bud/I Get a Kick Out of You.mid	Jazz
0.012	Mariah Carey/Can't Let Go.1.mid	Rhythm and Blues
0.016	Billy Joel/Root Beer Rag.mid	Ragtime
16	Basie/Have a Nice Day.mid	Jazz
	Acoustic Grand, Acoustic Bass, Alto Sax, and Brass	238
0.000	2 the Core/Have a Nice Day.mid	Jazz
0.013	Basie/Ya Gotta Try.mid	Jazz
0.014	Ellington/Satin Doll.1.mid	Jazz standard
0.014	Rich , Buddy/Groovin' Hard.mid	Jazz
0.017	Sammy Nestico/Pressure Cooker.mid	Jazz
0.019	Miller/In the Mood.1.mid	Big band
0.020	Parker Charlie/Scrapple from the Apple.mid	Jazz
0.021	Herman/'Round Midnight.mid	Bebop
0.023	Gillespie Dizzy/Salt Peanuts.mid	Jazz
0.025	Basie/Doin' Basie's Thing.mid	
0.026	Parker Charlie/Ornithology.mid	Bebop
0.029	Basie/Basie - Straight Ahead.mid	Big band

Table 2: Midi files associated with program clusters 15 and 16. The medoid file name is given in the first line with it probable genre. The next line lists the main programs which distinguish this cluster. The following lines list the r value (discrepancy) and midi file names of the best matches to the medoid.

r	file	genre
24	Def Leppard/Photograph.1.mid	Hard rock
	Distortion Guitar, Fretless Bass, others	298
0.000	Def Leppard/Photograph.mid	Hard rock
0.009	Gary Moore/Walking by Myself.1.mid	Hard rock
0.018	Def Leppard/Armageddon It.2.mid	Hard rock
0.032	Def Leppard/Heaven Is.mid	Glam metal
0.035	Nirvana/Son of a Gun.1.mid	Grunge
0.036	Jimi Hendrix/Foxy Lady.2.mid	Psychedelic rock
0.052	Iron Maiden/Holy Smoke.mid	Heavy metal music
0.056	Def Leppard/All I Want Is Everything.mid	Hard rock
0.057	Queen/Son and Daughter.1.mid	Hard rock
0.071	Green Day/When I Come Around.1.mid	
0.073	The Cult/Wild Flower.mid	

Table 3: Midi files associated with program cluster 24.

the same way the program vector was determined. The pitch class vector was normalized so that the total squared activity is one. For example, Figure 7 is a graphic representation of the pitch class vector for The Beatles song 'A Hard Day's Night'. The amount of activity in each of the pitch classes is indicated by the height of the black bar. F# is the only non natural note implying that the music is probably in the key of G major. Since G is the most active note, it is likely to be the tonal center providing further justification to assuming the music is played in G major.

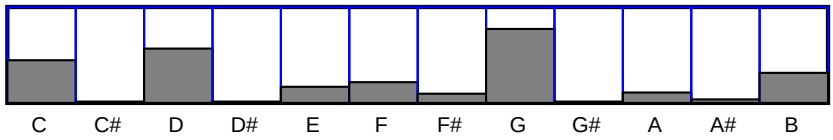

Figure 7: Pitch class vector for The Beatles song, 'A Hard Day's Night'.

The pitch class vectors can be quite varied. George Gershwin's 'Rhapsody in Blue' in Figure 8 appears to use a chromatic scale but in fact it still uses the diatonic scale. The music moves around different keys so much that it uses all the chromatic keys.

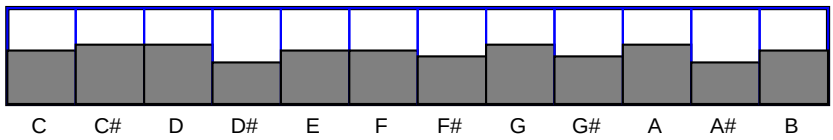

Figure 8: George Gershwin 'Rhapsody in Blue'.

In contrast, Darude's 'Sandstorm', seems to be limited to three pitch classes.

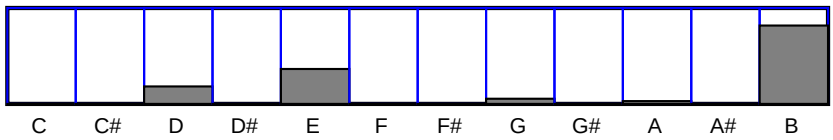

Figure 9: Pitch class distribution for Darude's Sandstorm.

A serious issue with this method is that the pitch class vector does not reflect the melody line which typically has more structure and detail. For example, Figure 10 was created using only the melody lines. Unfortunately there is no reliable automatic method for determining the MIDI tracks containing the melody lines [37] and [38].

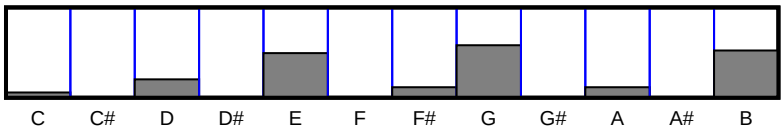

Figure 10: Pitch class distribution computed from the melody lines in Darude's Sandstorm.

The 17,200 pitch class vectors were partitioned into 30 groups using the clara algorithm. Except for cluster number 12 which tends to focus on country music and the blues, I did not detect any pattern in this partition based on pitch class information.

4 Analysis of Percussion Data

The MIDI standard assigns channel 9 exclusively to percussion instruments. For this channel, the pitch byte is not used to indicate the pitch of a musical note but instead specifies one of 47 percussion instruments. Nearly every midi file in the collection contains a percussion line which gives the music the appropriate ambiance.

As seen in Figure 11 a midi file can uses between zero and 35 percussion instruments. Certain percussion instruments such as the hi-hats are almost always present as seen in Figure 12

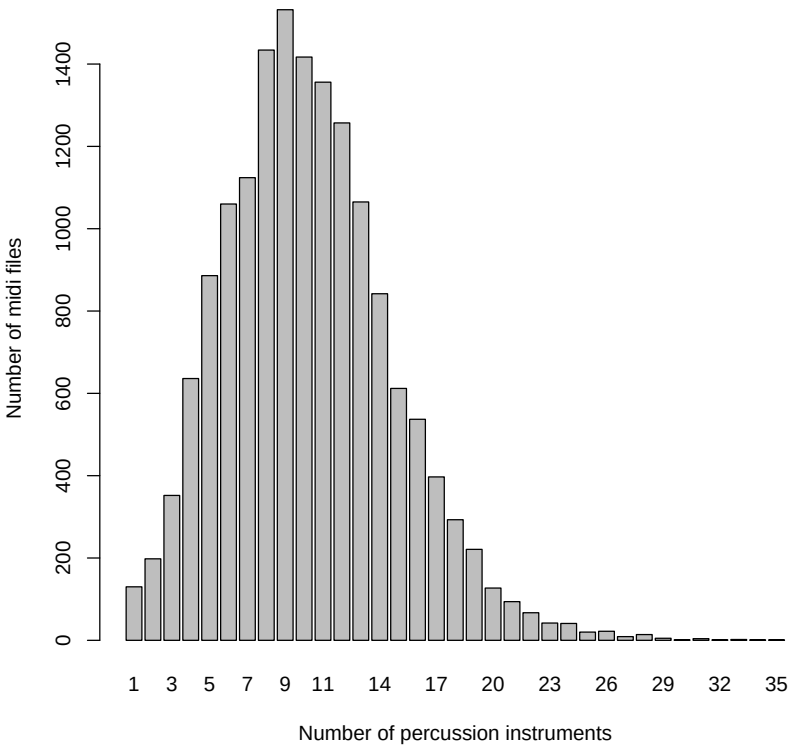

Figure 11: Histogram of the number of percussion instruments in a midi file.

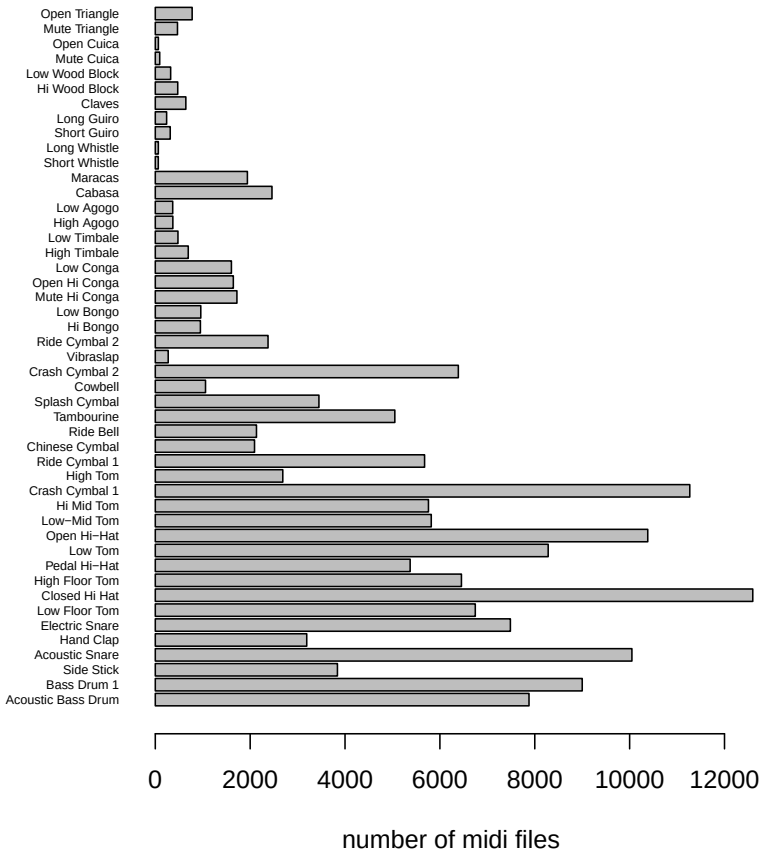

Figure 12: Histogram of the number of midi files which reference a particular percussion instrument is shown here.

In the case where five or more percussion instruments are playing simultaneously, it is unlikely that the listener is aware of all of the active instruments. Many of the instruments do not contribute to the overall beat and can be considered decorative rather than essential. It is likely that the midi files using 5 or few percussion instruments would concentrate on the essential percussion instruments. If we limit the analysis to those midi files that use five or fewer percussion instruments, we can compute a similar histogram as shown in 12. This collection represents about 20 percent of the entire midi file database. The fraction of the histogram counts for the limited midi file collection over the histogram counts for the entire collection would be indicative of the preference for certain percussion instruments in the limited set. This is plotted in 13. The set of midi files with five or fewer percussion instruments seem to avoid the Tom drums in favour for the bass drums, snare, and hi-hat cymbals.

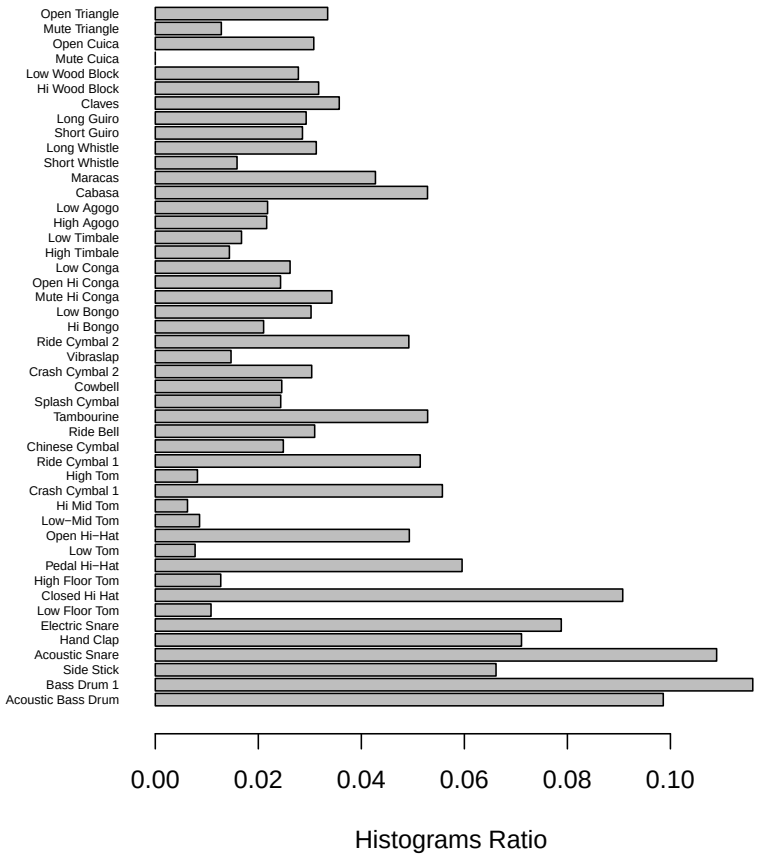

Figure 13: The ratio of the number of midi files which reference a particular percussion instrument in the set of midi files that contain 5 or fewer percussion instruments over the number of midi files for the entire database is plotted.

The nonnegative matrix factorization (nmf) [39] provides a formal method for extracting the natural grouping of the percussion instruments. This is one of several schemes for applying matrix decomposition to data mining [40]. nmf requires that all the components of the input vectors are never negative. It represents the data with a compact set of basis vectors.

The R statistical environment has two packages for computing the NMF. Package NMF [41] contains a large toolkit for computing the NMF using an assortment of algorithms as well as numerous functions for evaluating the resulting models. This package comes with 191 pages of documentation. Package NNLM [42] implements an efficent algorithm for the nmf which is suitable for large databases.

Let X represent the m by n data matrix where each of the m rows of the matrix represents a different attribute of the n objects. This is the transpose of the usual representation where the objects are aligned along the rows. The nmf computes the approximation

$$X \simeq WH \tag{2}$$

where we need to determine the positive valued matrices W and H. W is a matrix with m rows and r columns and H is a matrix with r rows and n columns where r is chosen to be less than n and m. W contains the basis vectors (factors) and H (mixing matrix) contains the coefficients to the basis vectors in the transformation for the different data vectors.

Unlike other matrix decomposition techniques such as the principle components and the singular value decomposition, the nonnegative matrix factorization does not return a unique solution. Different decompositions are obtained depending upon the chosen r size, the algorithm used, and the initial starting values.

In this study, a percussion vector listing the number of noteon commands for each of the 47 percussion instruments was determined for a particular midi file. The percussion vectors were normalized to a unit squared length. Percussion vectors were determined for each of the midi files and assembled into an input matrix X by appending all the percussion vectors as column vectors. The nmf decomposition with an r equal to 15 was applied to this matrix and the resulting basis vectors are shown in Table 4. Looking at the basis vectors consisting of the 15 columns in this table it is apparent that these basis vectors group similar types of percussion instruments.

The percussion accompaniment is open to deeper analysis. Generally, the percussion accompaniment consists of several loops of repeating patterns, where a pattern may extend over several beats. The onset times of the percussion notes are quantized to units of a 1/16 note. This is easily verified from the histogram of the onset times – for example Figures 14 and 15. The duration of a midi percussion note is meaningless, since the midi percussion hit always has a fixed decay.

Ignoring the loudness of the note, we can represent the sequence of notes of a particular percussion instrument as a binary time series of 1/16 notes. A value of one represents the hit of the percussion instrument at a particular time. Most of the other values in the series are zero, when the instrument is inactive. It is possible for two or more different percussion instruments to be hit at the same time. Therefore we represent the state of all the 47 midi percussion instruments at a particular instant by a binary vector of dimension 47. The percussion track is now represented by a time series of n binary vectors where n is the number of 1/16th notes in the entire midi file.

The series of binary vectors are best handled when they are represented by long integers. The binary operations such as 'exclusive or' and bit counting allow us to analyze these sequences efficiently. It is not too difficult to identify the various repeating patterns in the percussion track.

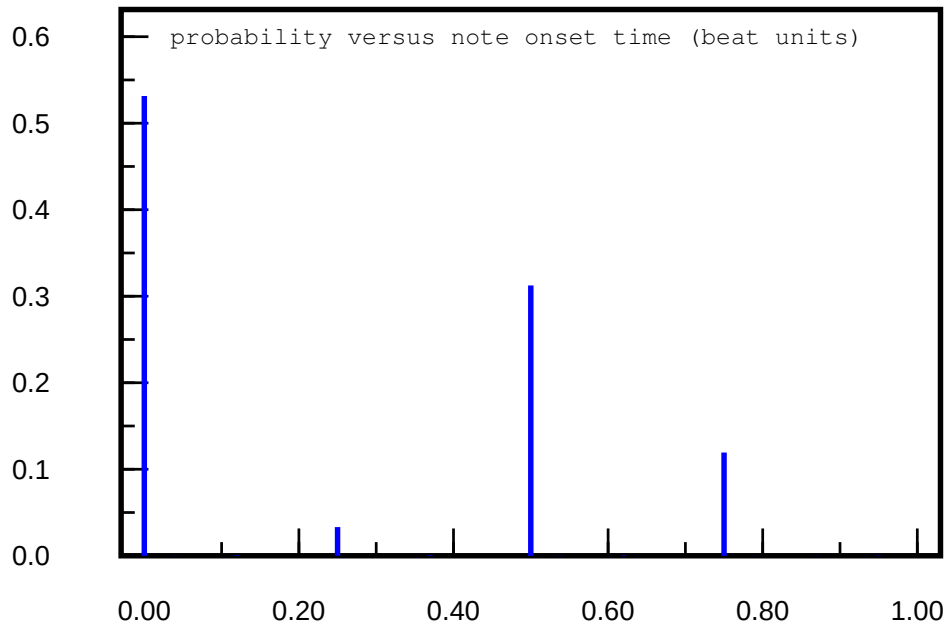

Figure 14: Probability of the percussion note onset versus time inside a beat. In this example all the percussion onsets are quantized to quarter beat units.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Acoustic Bass Drum	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.3	0.0	0.0
Bass Drum 1	0.0	0.0	0.0	0.0	16.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Side Stick	0.0	17.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Acoustic Snare	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.1	0.0	0.0	0.0	0.0
Hand Clap	0.5	0.5	0.5	1.6	0.4	0.1	0.0	0.0	0.0	0.1	0.5	0.1	0.3	0.4	0.0
Electric Snare	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.5	0.0	0.0	0.0	0.0	0.0
Low Floor Tom	0.0	0.1	0.0	0.0	0.4	0.0	0.0	0.0	0.7	0.1	0.3	0.0	0.4	0.0	0.1
Closed Hi Hat	0.0	0.4	0.0	0.0	0.0	0.0	0.0	10.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
High Floor Tom	0.0	0.0	0.0	0.1	0.3	0.0	0.0	0.0	0.6	0.4	0.2	0.0	0.2	0.0	0.3
Pedal Hi-Hat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	21.1	0.0
Low Tom	0.0	0.0	0.0	0.0	0.4	0.0	0.1	0.0	0.7	0.3	0.3	0.0	0.4	0.0	0.4
Open Hi-Hat	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.6	0.0	0.0	0.0
Low-Mid Tom	0.0	0.2	0.0	0.0	0.3	0.0	0.0	0.0	0.3	0.2	0.0	0.0	0.2	0.0	0.2
Hi Mid Tom	0.0	0.2	0.0	0.0	0.2	0.0	0.0	0.0	0.3	0.0	0.1	0.0	0.1	0.0	0.2
Crash Cymbal 1	0.0	0.0	0.0	0.0	0.6	0.0	0.3	0.0	0.4	0.2	0.6	0.4	0.5	0.0	0.0
High Tom	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.2	0.2	0.1	0.0	0.0	0.0	0.0
Ride Cymbal 1	0.0	0.0	0.0	0.0	0.0	0.0	11.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Chinese Cymbal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.2	0.1	0.0	0.0	0.0	0.0
Ride Bell	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.8	0.0	0.0	0.0	0.0	0.0	0.0
Tambourine	0.0	0.0	17.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Splash Cymbal	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.3	0.3	0.1	0.0	0.1	0.0	0.4
Cowbell	0.2	0.2	0.1	2.6	0.2	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.1	0.0
Crash Cymbal 2	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.4	0.3	0.2	0.2	0.2	0.0	1.2
Vibraslap	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
Ride Cymbal 2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	27.0
Hi Bongo	0.2	0.0	0.0	3.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3
Low Bongo	0.0	0.0	0.0	4.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
Mute Hi Conga	0.0	0.0	0.0	11.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Open Hi Conga	0.2	0.0	0.0	12.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Low Conga	0.0	0.0	0.0	11.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
High Timbale	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Low Timbale	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
High Agogo	0.1	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Low Agogo	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cabasa	0.0	0.0	0.0	0.0	0.0	12.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Maracas	22.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Short Whistle	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Long Whistle	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Short Guiro	0.2	0.3	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Long Guiro	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Claves	0.2	0.4	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Hi Wood Block	0.1	0.1	0.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Low Wood Block	0.0	0.1	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mute Cuica	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Open Cuica	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mute Triangle	0.3	0.3	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
Open Triangle	0.3	0.5	0.1	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2

Table 4: Nonnegative matrix factorization basis vectors (columns) for percussion lines for the midi file database.

number	class	percussion codes
1	Bass Drum	35, 36, 41
2	Snare	38, 40
3	Stick and Clap	37, 39
4	Hi Hat, Cymbal, Triangle	42, 44, 46, 51, 59 , 80, 81
5	Crash Cymbal	49, 52, 55, 57
6	Cowbell	56
7	Vibraslap	58
8	Bongo Conga	60, 61, 62, 62, 63, 64
9	Cabasa	9
10	Tom	41, 43, 45, 47, 48, 50
11	Tambourine	54
12	Agogo	67, 68
13	Ride Bell	53
14	Maracas	70
15	Whistle	71, 72
16	Wood Block	76, 77
17	Guiro	73, 74
18	Cuica	78, 79
19	Timbale	65, 66

Table 5: Groups of percussion instruments. The numbers in the last column refer to the percussion code number in the General Midi Standard.

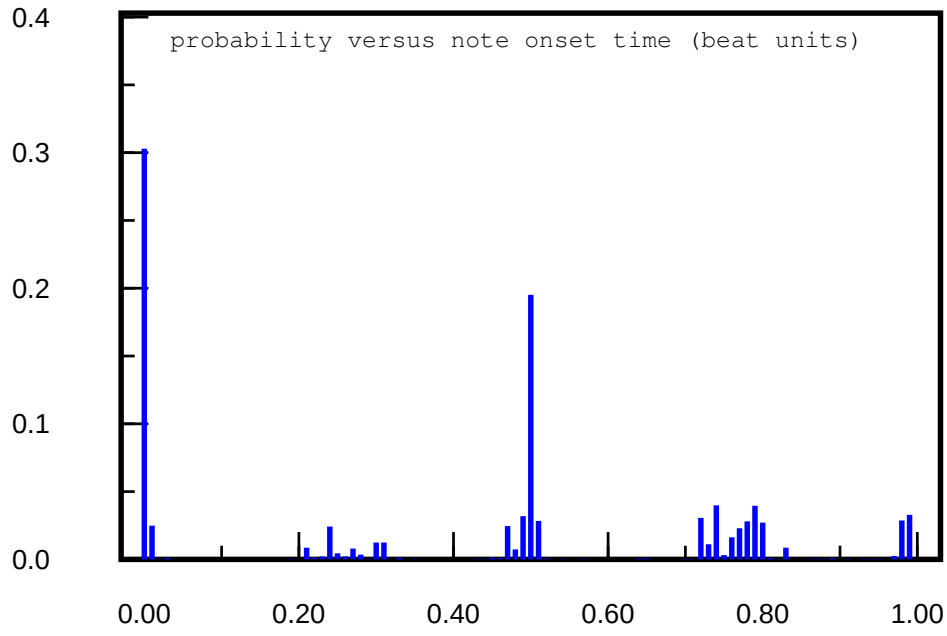

Figure 15: Probability of the percussion note onset versus time inside a beat. In this example not all the percussion onsets were quantized to quarter beat units.

In order to reduce the variability of the data, the various components of the binary percussion vectors were grouped. To do this, 47 percussion instruments were separated into 19 classes. For example, the Electric Snare is frequently substituted for Acoustic Snare in a particular midi file. This does not have much impact on the nature of the percussion accompaniment. Groupings that we used are shown in Table 5.

Each of the midi files references only a small number of the possible binary percussion vectors. The number of distinct binary percussion vectors in a midi file varies over a range of zero to little over a hundred as is evident in Figure 16. Each beat in the midi file, is formed by combining four of the sequential binary vectors. The number of distinct percussion beats can also be determined and is shown in Figure 17.

As mentioned earlier, the percussion line tends to be composed of repeating percussion sequences of one or more beats. Assuming all of these sequences are the same length, the periodicity of the the percussion line can be determined from the autocorrelation function of the binary vector time series. Figure 18 shows a sample autocorrelation function for one of the midi files. Limiting the search to the maximum offset of 16 beats, the highest peak suggests a periodicity of four beats. This implies that most of the repeating patterns are four beats long. An analysis of the entire collection of midi files, indicates that this is the most common length of the repeating patterns – Figure 19.

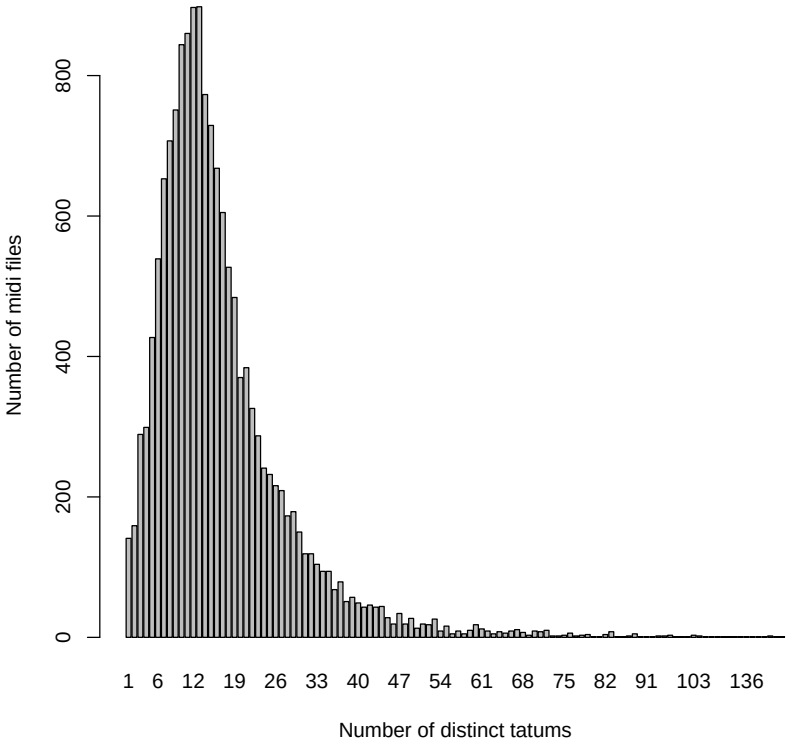

Figure 16: Probability of n distinct binary percussion vectors appearing in a midi file.

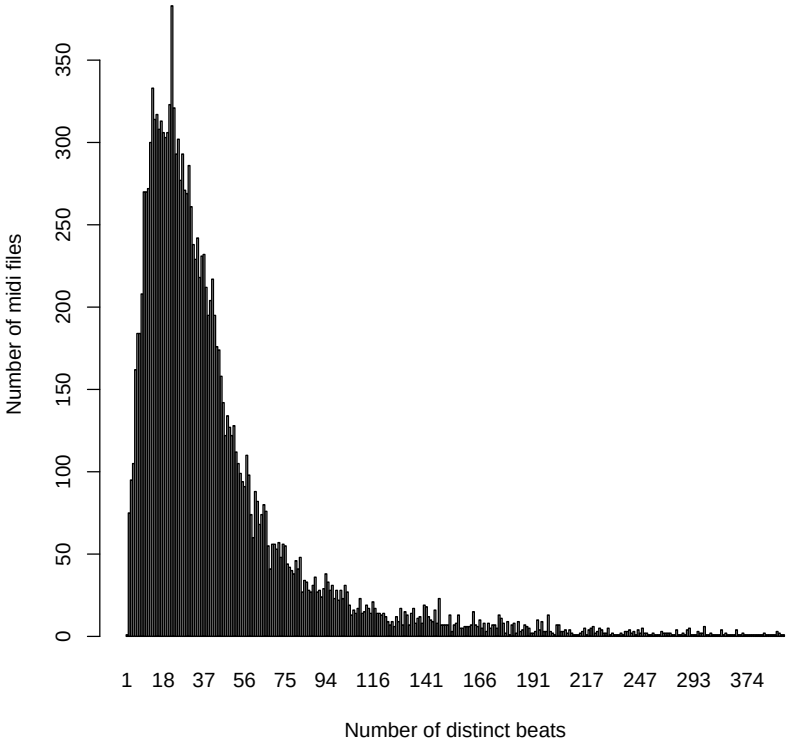

Figure 17: Probability of n distinct percussion beats appearing in a midi file.

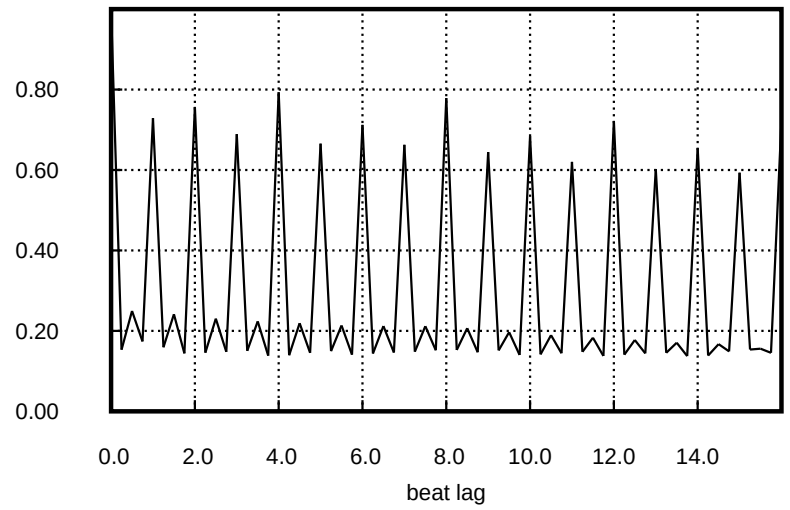

Figure 18: An example of the autocorrelation function of the percussion binary vector time series. The function implies a periodicity of 4 beats.

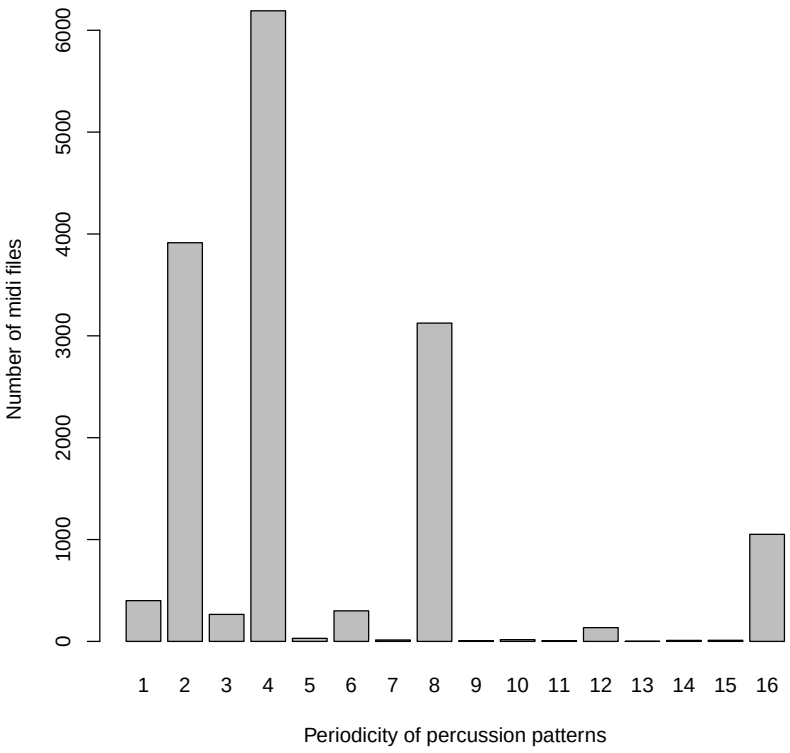

Figure 19: Histogram of percussion pattern periodicity for the collection of midi files.

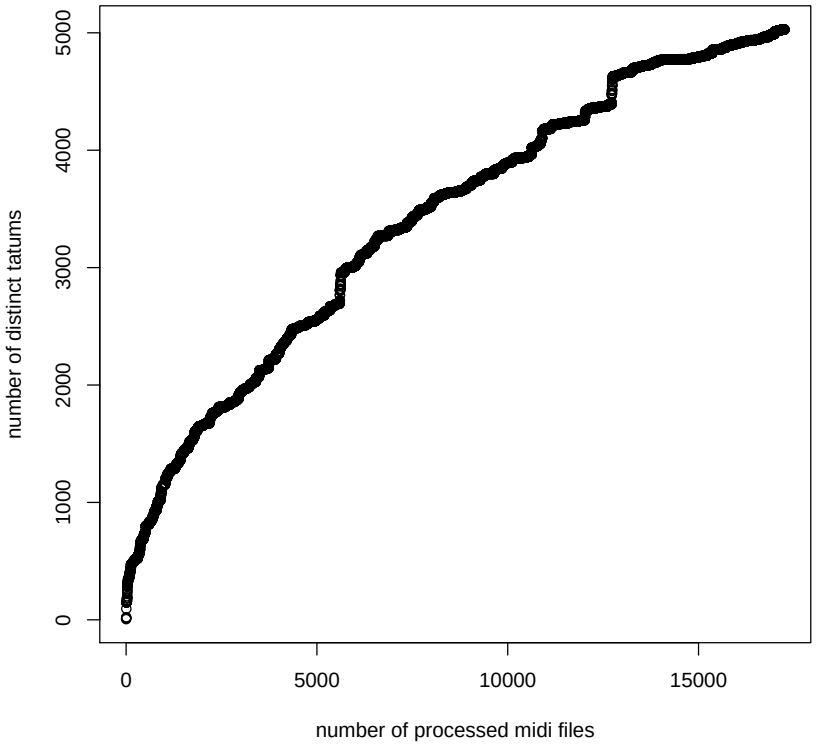

Figure 20: Number of distinct discovered tatums versus number of midi files analyzed.

5 Conclusion

The application of these algorithms on real data is not a trivial matter [34]. First there is the question of what is a true cluster [43]. Then there is the question of which clustering algorithm is appropriate for the type of data. Clustering algorithms usually require the user to specify the number of clusters that are expected and perhaps other parameters which effect the processing time and accuracy of the results. There is no guarantee that the analysis will return meaningful results.

Hennig [44] lists 7 decisions that are required in any cluster analysis. They are 1) choosing the objects to be clustered, 2) choosing the measurements/variables, 3) standardization of the variables, 4) choosing a similarity measure, 5) choosing a clustering method, 6) determining or deciding the number of clusters, and 7) interpreting and validating the results. All of these decisions apply here.

References

[1] Colin Raffel and Daniel P.W. Ellis. Extracting ground-truth information from midi files: A midifesto. In *ISMIR 2016 Proceedings of the International Society for Music Information Retrieval*, pages 796–802, 2016.

[2] *International MIDI Association. General MIDI level I specification 1991.*

[3] Colin Raffel. *The Lakh MIDI Dataset v0.1*, 2016. `http://colinraffel.com/projects/lmd/`.

[4] *Million Song Dataset*. `http://labrosa.ee.columbia.edu/millionsong/`.

[5] Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD dissertation, 2016.

[6] *List of popular music genres*, 2018. https://en.wikipedia.org/wiki/List_of_popular_music_genres.

[7] Glenn McDonald. *Every Noise at Once*, 2013. http://everynoise.com/engenremap.html.

[8] Cory McKay. *Automatic Music Classification with jMIR*. PhD dissertation, 2010.

[9] Marcelo G. Armentano, Walter A. De Noni, and Herman F. Cardoso. Genre classification of symbolic pieces of music. *Journal of Intelligent Information Systems*, 48(3):579–599, 2017.

[10] David Huron. *The Humdrum Toolkit: Software for Music Research*. http://www.humdrum.org/.

[11] Michael Scott Cuthbert annd Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. In *ISMIR 2010 Proceedings of the International Society for Music Information Retrieval*, pages 637–642, 2010.

[12] Seymour Shlien. *Midiexplorer*, 2017. https://sourceforge.net/projects/midiexplorer/.

[13] Seymour Shlien. *MidiExplorer*, 2017. https://midiexplorer.sourceforge.io/.

[14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. http://www.R-project.org/.

[15] Cory McKay and Ichiro Fujinaga. Improving automatic music classification performance by extracting features from different types of data. In *Proceedings of the ACM SIGMM International Conference on Multimedia Information Retrieval*, pages 257–266, 2010.

[16] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R, 7th Edition*. Springer, 2017.

[17] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[18] Michael Hahsler, Matthew Piekenbrock, and Derek Doran. *dbscan: Fast Density-based Clustering with R*. https://cran.r-project.org/web/packages/dbscan/vignettes/dbscan.pdf.

[19] Marzena Kryszkiewicz and Lukasz Skonieczny. Faster clustering with dbscan. In *Intelligent Information and Web Mining. Advances in Soft Computing*, volume 31, pages 605–614. Springer.

[20] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Joerg Sander. Optics: Ordering points to identify clustering structure. In *ACM SIGMOD International Conference on Management of Data*, pages 49–60. ACM Press, 1999.

[21] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? 1540:217–235, 12 1998.

[22] Frank Klawonn, Frank Hoppner, and Balasubramaniam Jayaram. What are clusters in high dimensions and are they difficult to find? In *Revised Selected Papers of the First International Workshop on Clustering High–Dimensional Data*, volume 7627, pages 14–33, 2012.

[23] Michael Steinbach, Levent Ertoz, and Vipin Kumar. *The Challenges of Clustering High Dimensional Data*, 2014. https://www-users.cs.umn.edu/~kumar001/papers/high_dim_clustering_19.pdf.

[24] S.P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1957.

[25] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., 2018.

[26] Preeti Arora, Dr. Deepali, and Shipra Varshney. Analysis of k-means and k-medoids algorithm for big data. *Procedia Comuter Science*, 78:507–512, 2015.

[27] Alboukadel Kassambara. *Determining the Optimal Number Of Clusters: 3 Must Know Methods*, 2017. www.sthda.com/english/articles/29-cluster-validation-essentials/96-determining-the-optimal-number-of-clusters-3-must-know-methods/.

[28] Christian Hennig. *Flexible Procedures for Clustering*, 2018. ftp://cran.r-project.org/pub/R/web/packages/fpc/fpc.pdf.

[29] Malika Charrad, Nadia Ghazzali, Veronique Boiteau, and Azam Niknafs. *Determining the Best Number of Clusters in a Data Set*, 2015. https://cran.r-project.org/web/packages/NbClust/NbClust.pdf.

[30] Mark Ming-Tso Chiang and Boris Mirkin. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of Classification*, 27:3–40, 2010.

[31] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, Junjie Wu, and Sen Wu. Understanding and enhancement of internal clustering validation measures. *IEEE Transactions on Cybernetics*, 43(3):982–993, 2013.

[32] Alboukadel Kassambara. *Statistical tools for high-throughput data analysis*, 2017. http://www.sthda.com/english/.

[33] Phil Spector. *Cluster Analysis*, 2011. Class notes for Stat133, www.stat.berkeley.edu/~s133/Cluster2a.html.

[34] Ala Al-Fuqaha. *Partitional (K-means), Hierarchical, Density-Based (DBSCAN) Clustering Analysis*. CS 6530: Data Mining (Summer 2014) https://cs.wmich.edu/~alfuqaha/summer14/cs6530/lectures/ClusteringAnalysis.pdf.

[35] *Package 'cluster'*, 2018. https://cran.r-project.org/web/packages/cluster/cluster.pdf.

[36] *Jazz scales*, 2018. https://en.wikipedia.org/wiki/Jazz_scale.

[37] David Rizo, Pedro J. Ponce de Leon, Antonio Pertusa, and Jose M. Inesta. Melodic track identification in midi files. In *IbPRIA 2009: Pattern Recognition and Image Analysis*, pages 489–496, 2009.

[38] Soren Tjagvad Madsen and Gerhard Widmer. A complexity-based approach to melody track identification in midi files. In *Proceedings of the International Music Conference (ICMC'07)'*, 2007.

[39] N. Gillis. The why and how of nonnegative matrix factorization". In *Regularization, Optimization, Kernels, and Support Vector Machines*, pages 257–291, 2014.

[40] David Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Springer-Verlag, 2007.

[41] Renaud Gaujoux and Cathal Seoighe. *Algorithms and Framework for Nonnegative Matrix Factorization (NMF)*, 2018. https://cran.r-project.org/web/packages/NMF/index.html.

[42] Xihui Lin and Paul C Boutros. *NNLM: Fast and Versatile Non-Negative Matrix Factorization*, 2018. https://cran.r-project.org/web/packages/NNLM/index.html.

[43] Christian Hennig. What are the true clusters? *Pattern Recognition Letters*, 64:53–62, 2015.

[44] Christian Hennig. Clustering strategy and method selection. *Arxiv.org*, 2015. https://arxiv.org/abs/1503.02059.