Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-018-0321-2

ORIGINAL PAPER

Music classification as a new approach for malware detection

Mehrdad Farrokhmanesh[1] · **Ali Hamzeh**[1]

Received: 20 February 2017 / Accepted: 24 April 2018

Abstract
Each year, a huge number of malicious programs are released which causes malware detection to become a critical task in computer security. Antiviruses use various methods for detecting malware, such as signature-based and heuristic-based techniques. Polymorphic and metamorphic malwares employ obfuscation techniques to bypass traditional detection methods used by antiviruses. Recently, the number of these malware has increased dramatically. Most of the previously proposed methods to detect malware are based on high-level features such as opcodes, function calls or program's control flow graph (CFG). Due to new obfuscation techniques, extracting high-level features is tough, fallible and time-consuming; hence approaches using program's bytes are quicker and more accurate. In this paper, a novel byte-level method for detecting malware by audio signal processing techniques is presented. In our proposed method, program's bytes are converted to a meaningful audio signal, then Music Information Retrieval (MIR) techniques are employed to construct a machine learning music classification model from audio signals to detect new and unseen instances. Experiments evaluate the influence of different strategies converting bytes to audio signals and the effectiveness of the method.

Keywords Malware detection · Music Information Retrieval (MIR) · Audio signal processing · MFCC · Classification

1 Introduction

Malware means 'malicious' 'software' and is used as a general term for any software designed to purposely damage, steal user's information or do other unwanted actions [1]. Nowadays, computer and the internet play an essential role in every day's life and everyone performs almost all of their daily tasks by the Internet such as online banking, purchasing, etc. [2]. Just as in the physical world, there are people on the Internet with malevolent intents that strive to enrich themselves by taking advantage of legitimate users whenever money is involved. Malware (i.e., software of malicious intent) helps these people accomplishing their goals [2].

To protect legitimate users from the mentioned threats, security vendors offer tools for identifying and coping with malicious software by applying different malware detection techniques. The most widely used one was signature-based in the previous years. In signature-based approaches, a signature has been extracted from a propagated malware file which is typically an exclusive substring of bytes. Then, this signature has been compared to the signature database in order to find the match [3]. Although signature-based techniques are rapid and precise, modifying malware by obfuscation methods can thwart the anti-virus easily and make the detection impossible. Today, a great number of heuristic approaches have been suggested to solve the obfuscated malware detection problem. In this paper, we use the terms anti-virus and anti-malware interchangeably. Code obfuscation is a technique that changes the structure of an executable program while its functionality left unchanged [3]. Obfuscation tries to alter a malware in a way that traditional anti-virus programs consider them as a benign file, so that detecting obfuscated malware become a challenging task.

To cope with obfuscation techniques, anti-viruses apply new detecting approaches that are divided into two main categories: static and dynamic analysis [3]. Static analysis techniques utilize file's byte, machine language instructions, program's opcodes, program flow, function names, headers or all the other items that are not related to the behavioral features of the file. However, dynamic analysis approaches consider the program's behavior attribute like resource consumption pattern or generated network traffic in execution time. Although dynamic analysis provides better information

✉ Ali Hamzeh
ali@cse.shirazu.ac.ir

Mehrdad Farrokhmanesh
farrokhmanesh@cse.shirazu.ac.ir

[1] Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

Published online: 21 May 2018

about files, static analysis used more due to its simplicity. In addition, dynamic analysis methods are time consuming and in some cases smart malware may become aware that they are under investigation and try not to uncover themselves.

There are two main type of obfuscation: Polymorphism and Metamorphism. A polymorphic malware is a kind of malware that its malicious content has been encrypted by a polymorphic generator (packer) [3]. Polymorphic malware contains a decryptor engine that decrypts the malicious code on the fly and jump to malicious code entry point, therefore dynamic analysis can be reveal them easily. On the other hand, static approaches usually are not capable of detecting these types of malware.

Due to the mentioned weaknesses, malware developers find out new techniques called "metamorphism" which is changed the malicious code's structure instead of encrypting it [4]. Metamorphic malware, automatically recodes itself in each propagation by performing different methods such as adding various length NOP instructions, adding useless instructions, loops and conditions within the code, function reordering, program flow modification, static data structure modification, permuting registers in the instructions and so on. [4].

As another categorization, we can perform static analysis in a high-level or low-level manner. When we consider a file as a sequence of bytes, we use byte-level static analysis [3]. On the other hand, when we interpret these bytes into information such as opcodes (operational part of an instruction code), API calls, control flow graphs (CFG), export table and etc., we do high-level file analysis which is more accurate if the retrieved information is corrected but providing this information requires a large amount of time, more resources and also in some conditions due to some reasons accessing demanded data is not feasible and error-prone. Recent byte-level approaches show that even without this extra effort, we can get promising results.

In this paper, we propose a static method for detecting malware using raw bytes for extracting features by the traditional audio signal processing techniques. Music Information Retrieval (MIR) is the science of retrieving information from music [6]. One of the most widely used application of MIR is automatic music classification which is a supervised approach that can be done based on the music content, genre, artist, the place of origin or the mood of the piece and assign a label to each of the music by the previous views [7]. In the mentioned music classification task, our goal is to find similar musical patterns in audio signal. We used audio-specific features such as MFCC [8] and chromagram [17] to classify music. Previous works depicted that efficient machine learning models can be trained by these features to classify music accurately.

In our method, we provide an approach for malware classification by using the similarity between the essence of music structure and executable file. This technique considers malware detection problem as a music classification and try to detect malware by employing music detection algorithms.

It is noticeable that, in the malware classification methods which are based on machine learning techniques, the most indispensable challenge is to trigger an impressive algorithm to extract features from executable files that can be a good representation of the file's content. Our goal is to demonstrate that, by converting an executable file into a music, our proposed method can detect malware using the algorithms of music classification. We do not claim that, our approach is the best option to detect malware however regarding the results, we depict that by applying the algorithms in the field of music classification, pleasant results are achieved without facing the problem of finding an efficient algorithm. The superiority of our approach is as follows:

- It is a byte-level static approach so; it doesn't need to disassemble the file or extract function calling graph or finding API calls which is time-consuming and error-prone.
- It can detect label of the file with only analyzing a portion of the file.
- It is simpler than most of the machine learning approaches while being precise and fast.
- It is lightweight; hence it can be used in production environments.

The remainder of this paper is organized as follows. In Section 2 we discuss related work for static malware detection both in byte level and opcode level. Section 3 provides background information about digital music production and classification. Section 4 consider our method to detect unknown malware by using MIR techniques. In Section 5, experimental results for evaluating our method and comparing to other relevant approaches are provided. Section 6 contains our conclusion and a discussion of possible future work.

2 Related work

In this section, some of the related detection methods which are based on bytes and other extracted features from files will be provided.

Y.Zou et al. [9] introduced a byte-level method based on compression techniques and applied Prediction by Partial Matching (PPM) that is also called Prediction by Markov Model of Order 'n'. PPM is a data compression technique that uses context modeling and prediction and is also employed to estimate the next symbol from 'n' previous symbols. Y.Zou's method utilizes PPM as a classifier for classifying unknown files. In that models, two PPM models are trained,

one for the benign set and the other for malware set. Then, an unknown instance is compressed using both models and the PPM model which can compress the file better (can more reduce its size) indicates its label. The philosophy behind the compression based methods is the fact that a file can be better compressed, if it contains more substrings similar to the seen substrings in the trained model. The main drawback of compression-based classifiers like PPM, is that they need a huge amount of memory to maintain compression models which become more voluminous by increasing the number of files, thus the number of files for training the model is limited. Also, the required memory to maintain the model enhances dramatically as the order of the model (which is the number of previous symbols used to estimate each mode) increases. To deal with memory complexity, in Y.Zou's method, the PE header of files is ignored and the first block of the file is used after omitting the null bytes. It is remarkable that, determining the optimal size of the block is a complex task which choose to be 3072 bytes in Y.Zou's technique. In addition to the mentioned memory problem, compression based approaches are time-consuming.

Khorsand et al. [10] improved the Y.Zou's method [9] by choosing the different part of the file intelligently which is applied for creating models. It is considerable that, in some cases the malware can consist of the benign content in the first block which leads to confusing the detection system. Owing to the fact that, the first block after header cannot represent all content, they used PE headers and ASCII strings in the file for creating PPM models. Regarding the smaller size of PE header compared to larger block of code section, the memory requirement will be decreased in their method. Another approach employed in the Khorsand's method is extracting the ASCII strings with at least 4 bytes' length from the file for training the PPM. They depicted that their method is more accurate than Y.Zou's technique while maintaining PPM models are still memory consuming. Although they can achieve better accuracy by using the higher order for training PPM models, it needs a lot of memory and it is slow. Therefore, it cannot be employed in production world.

Schultz et al. [11] provided the byte-level malware detection by machine learning approaches. The method divides the program into non-overlapping byte substrings and create a feature vector based on the frequency of the substrings appear in a file. Then, use the features to train a data-mining model. The method is tested on several parts of the files like PE headers, string sequences or a piece of the program bytes. This method does not have the acceptable performace because it does not consider the overlap in byte sequences.

Kolter et al. [12] improved Schultz's method [11]. In the Kolter's proposed method, instead of using non-overlapping byte substrings, it uses n-grams that are overlapping substrings. In this approach, all n-grams of the file are taken and considered as a Boolean attribute in feature vector that is True if the n-gram is exist in the file or False if the n-gram is absent from the file, so that each file is represented as a Boolean vector. Next, the method computed the Information Gain (IG) for each n-gram and select the most 500 relevant n-grams. Then deliver the modified feature vectors to a classifier for training a machine-learning model such as SVM, decision tree or Naïve Bayes.

In addition to the above, some methods are based on signal processing techniques for detecting malicious software, similar to our work [13–15]. These approaches attempt to visualize (2D or 3D) malware contents (bytes or opcodes) or behavior logs of malware activities. Then, image processing techniques are applied to detect malware and generate an image from each file in the dataset.

Some image-based techniques similar to methods in [13] and [14] work on bytes which directly map the bytes or the respective entropy value of them to an image and apply simple image processing approaches to obtain similarity between images. Hence, an unknown file that is alike to a malware image is assigned a label as a malware instance.

Nataraj et al. [15] provide an image based technique that works on bytes. Unlike the previous mentioned techniques, they extract several texture features from generated images, for classification. The principal advantage of the method versus previous image-based methods is that it employs textures from images and uses classification which can increases accuracy to detect malware.

Image is a 2-dimensional signal, while audio signal is a 1-dimensional signal. Audio processing techniques are more simple, speedy and economical in resource consumption than 2-D image approaches. In addition, defined value for image height and width affect the final accuracy which is considered as a problem [15].

Santos et al. [5] proposed a method based on opcode of the executable files. First, files are disassembled and opcodes are extracted and then opcode profile that is a matrix shows the frequency number of each opcodes in all files is generated. Then, the relevance of each opcode is computed based on the frequency of their presence in each class (malware and benign) by statistical dependence of the opcode and the class. They consider n-gram opcode substrings as a term and calculate the term frequency (TF) of each substring and also defined Weighted Term Frequency (WTF) as the result of weighting the TF with the relevance of each opcode. WTF is computed as the product of sequence frequency and the calculated weight of every opcode in the sequence. A vector is created for each file that each of its elements represent the WTF related to an opcode sequence. Finally, these vectors are utilized to train a machine-learning model by the purpose of classifying unknown instances. Santos method has some disadvantages such as the need of disassembling file which causes extra processing step that is error-prone and time-consuming and also the impossibility of achieving

all opcodes when some anti-disassembling techniques were used. In addition, each instruction code is consisting of two parts, operational codes (opcode) and instruction operators where Santos method ignores instruction operators and leads to the lower accuracy and higher false alarms. It is significant to remark that, finding the optimal length of n-gram opcode sequences is a critical issue, a small sequence will fail to detect complex malicious blocks whereas long substrings can be avoided by simple obfuscation techniques in metamorphic malware.

Hashemi et al. [16] provided an opcode-based machine-learning approach which uses graph embedding techniques for unknown malware detection. In the first step which named graph extraction, the executables are disassembled to obtain their opcodes, then a directed graph from the opcodes of each sample is created. Each opcode in the program considered as a node in the graph. For successor opcodes in the program, a directed edge is inserted from the first opcode node to the successor opcode node. Edge weights indicate the probability of appearance of successor nodes (2 grams). The second step is graph embedding. In order to use machine learning algorithms which, apply vectors as the input, graph embedding is employed for mapping the graph to a vector, thus the most valuable eigenvector of the graph should be found. Eigenvectors and eigenvalues are powerful properties of a graph that represent the directions of nodes' distribution by considering the structure of the edges. Each graph can be representing by a set of eigenvectors where each eigenvector has an eigenvalue that indicates the level of importance. Power iteration is an iterative method for finding the most substantial eigenvector of the graph. In each iterate, it finds an eigenvector and the corresponding eigenvalue and iterates until the new eigenvalue has not an effectual difference from previous eigenvalue. Each sample is demonstrated as an eigenvector. The final calculated eigenvectors are used as representative sample feature-set for training a machine learning model. In addition to the mentioned problems related to opcode based methods, this approach demands a large number of iterations to converge eigenvectors into the most useful eigenvector which reduces the speed of the method. Also, with fewer iterates, the resulting features may not be very informative.

3 Background

In this section, we briefly discuss the basic concepts of the topics which is mentioned in this paper: Music genre-based and content-based classification task, The MIDI standard and producing digital audio by computer and the most used techniques for audio features extraction in Music Information Retrieval (MIR) field.

3.1 Music classification

Music Information Retrieval (MIR) is the science of retrieving information from music [6]. It is divided into two main categories, information retrieval based on the music content (melodies, genres etc.) and information retrieval based on the music metadata (cover image, artist, labeled genre etc.) [6]. Our discussion in this article is based on information extracted from the content of the music. One of the challenging tasks in the MIR field is automatic music classification based on genre. To do this, we need to employ audio signal processing techniques such as MFCC [8] and chromagram [17]. The techniques work on frequency domain. So, the audio signal that is in time domain is divided into equal frames with some overlap in successive frames. Then a frequency-based audio feature extraction technique such as MFCC is done on the frames. Finally, these feature sets can be used to train a machine learning classifier like K-Nearest Neighbors (KNN) [18] or Support Vector Machine (SVM) [18].

3.2 Musical instrument digital interface (MIDI) Files

The standard protocol to produce digital music in computer is MIDI [26]. MIDI is an event-based system. MIDI's events are transacted in the form of MIDI messages. MIDI event messages can be received from a physical or software musical instrument in real time or can be read from a MIDI file [27]. Generating digital music in MIDI standard is a quick, attractive and precise task.

MIDI messages determine what and when a musical note is played, the sound loudness (volume), the duration of the notes and also the instrument played this note [27].

An intricate music is constructed by combining the diverse sounds of instruments. In order to be capable of producing complex music, MIDI defines channels. Each MIDI message is related to a specific channel and each channel consists of independent sound from the other. Channels have unique characteristics and play separate instrument, therefore various notes with distinguished attributes and disparate instruments in separate channels can be created. Multiple channels can be played at the same time to play complex music. MIDI support up to 16 channels.

MIDI channels carry different MIDI messages. On the other hand, there is another notion of output channels in digital music. Audio output channels are related to the audio files and each channel bearing a diverse audio signal which makes the output sound more spatial (stereo). Accordingly, the concept of MIDI channels and output channels are completely different [27].

To determine the beat in each MIDI channel, the tempo time factor is used. The tempo specifies the number of time units per minute, and its metric is beat per minute (bpm).

The tension of a note in MIDI is assessed with the tempo value [27]. For example, if the tempo is 120 bpm (that is, each unit is 0.5 sec), and the duration of a note is 1 unit, this note played for 0.5 seconds.

The device is used for generating sound from MIDI messages is called synthesizer [26]. There are two type of synthesizer, hardware synthesizers and software synthesizer. A software synthesizer listens to the MIDI events and generate the corresponding sound by a SoundFont. SoundFont is a file that contains sample synthesizes for playing different musical instruments and sound effects by a computer.

To produce the audio signal, synthesizer generates a number of samples per second. The sampling rate is measured by Hz. The data of each digital sample is stored in a number of bits. The bitrate specifies the number of bits per second and its unit is bit per second. The higher the sampling rate and the bitrate, the higher the quality of the produced signal [26].

3.3 Chromagram

Chromagram or harmonic pitch class profile is a feature set used for automatic chord recognition [17]. A musical chord is a set of simultaneous tones and a subsequence of chords over time is called harmony. Automatic chord recognition is useful for mid-level musical signal representing in order to do harmonic analysis, music segmentation, music similarity measurement, audio summarization and other music information retrieval tasks.

Music pitches are two dimensions. The first dimension is the height that indicates the octave of the note, and the second is Chroma that represents the position of the note in relation to other notes which are in the same octave. A chromagram is a 12-demintional vector and each dimension in this vector demonstrate the intensity of a semitone in a chromatic scale. In this technique, a chord is determined by the position of tones in a Chroma, and their octave is ignored. Original tones are C, C#, D, D#, E, F, F#, G, G#, A, A# and B.

The procedure of making a chromagram from a piece of music is discussed in the following:

Step 1: Making the spectrogram
A spectrogram is a visual representation of the signal spectrum frequencies over time. A bin in spectrogram indicates the intensity of each frequency at the related time. Spectrogram frequency range is in 0 Hz to the Nyquist rate of the original signal [17].

Step 2: Converting frequencies to chromatic scale
At first, we should decide how to quantize the Chroma range which is quantized into 12 pitch classes by default (semitones). Next, we should find the chroma for each frequency in a time step. Chromagram is a many to one function that maps several related frequencies to one Chroma class.

Step 3: Creating Chroma bins
The purpose of this step is to map each frequency to a 12-state chroma. The aggregation of the amplitude of the frequencies that mapped to the same chroma class is calculated that indicates the intensity of each chroma for the time step. Then, we can visualize them, in each time step we have 12 bins that each bin represents the intensity of its Chroma.

3.4 Mel-frequency cepstral coefficients (MFCC)

The most popular and widespread technique to extract information from an audio signal is calculating its Mel-Frequency Cepstral Coefficients (MFCC) [8]. These features are used in speech recognition, speaker recognition and Music Information Retrieval (MIR) tasks [8]. MFCCs are based on frequency domain but they are computed in Mel scale which is based on human auditory system. Cepstral coefficients in Mel-scale is a representation of the original cepstral from a short window of signal that derived from the Fast Fourier Transform of the signal [8]. Unlike real cepstral, in Mel-scale cepstral, a nonlinear frequency scale is used similar to human ear system. These features are robust against noise and recording conditions and speakers. MFCCs are computed in the following steps:

Step 1: Framing and windowing the signal
Signal is divided into equal frames, overlapping on successive frames is also used. If the start and the end of a finite frame sample don't match, then the signal has discontinuities when we apply FFT on the frame. For eliminating such discontinuity, a windowing function like Hamming window [8] is employed to match up the ends. Hamming window filter coefficients are calculated according to the equation 1 [8]:

$$W(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right)$$
$$\text{With } \alpha = 0.54 \text{ and } \beta = 1 - \alpha = 0.46 \quad (1)$$

where N is the total number of frames and n is the current frame.

Step 2: Computing Fast Fourier Transform and applying Mel-scaled filter bank
A band-pass filter is a device that passes frequencies within a certain range and rejects frequencies outside that range. In signal processing, a filter bank is an array of band-pass filters that separates the input signal into multiple components, each one carrying a single frequency sub-band of the original signal.

Mel-scaled filter banks are a collection of some triangular bank filters that gives diffent energies to the diverse frequencies of the original signal based on the Mel scale. This causes the elimination of some frequencies, makes some of them more essential and some of them less momentous and also transferring the genuine signal to the signal in Mel scale domain. In the Mel-scale filter bank, bandwidth of higher frequency filter is larger than the lower frequency filters which are in temporal view seem to be similar.

In this step, Fast Fourier Transformation (FFT) of each frame is calculated which gives us frequency components of the signal in the time domain. Then, Mel-scale filter bank is applied on the given FFT. Mel scale is logarithmic and approximately linear up to 1 KHz. The relation between real frequency and Mel-scale frequency is formulated in the following [8]:

$$MelScale\ f = \left[\frac{2595\log\left(1 + f\ (Hz)\right)}{700}\right] \qquad (2)$$

Step 3: Discrete Cosine Transform

In this step, the Discrete Cosine Transformation (DCT) of the filtered FFT from previous section is computed. DCT provide coefficients that arranged by their importance [8]. The result of this step is called Mel Frequency Cepstrum Coefficients (MFCCs). A vector consisting of these coefficients is called acoustic vector. Typically, we obtain 12 coefficients for each frame.

4 The proposed method

In this paper, by using the similarity between an executable file and an instrumental music, we provide a method for solving the problem of malware detection by the help of techniques in the field of music classification.

As stated before, music is more structural than other types of audio (speech or nature voices). A sequence of building blocks represents a great resemblance in music and executable files which in music consist of musical notes and in executable file is bytes. Musical patterns or rhythms generate by the number of notes with certain order and the existence of them can help the audio classification by genre significantly, whereas in executable files byte sequences is valuable item. It is noticeable that, generating some byte patterns that abundantly exist in malware files is a lucrative side effect of metamorphic generators and they can be used for detecting malicious files which is cited in some of the previous works.

In our proposed technique, by the aim of making method plain, fast and more beneficial, we consider executable files as a sequence of bytes regarding the availability of bytes without any extra processing. Indeed, bytes are always attainable compared to opcodes which cannot achieve in some situations like when anti-disassembling techniques are used [3].

Our approach is based on mapping the mentioned problem to different field with a simple technique, the challenge of finding the effective algorithm for feature extraction which is the hardest step in heuristic malware detection methods is omitted. Therefore, our approach does not have the claim to be the foremost way for malware detection and be more practical than methods that are used bytes directly and do not maps bytes to frequency domain. Although we are not assert that our approach is the only available way for transforming the executable files to audio (specifically music), we demonstrate that in the most straightforward condition we gain acceptable results with this method.

In the following, the explanation of each step of our method and discussion of different challenges will be represented in detail. Our suggested method has five main steps: converting executable file to MIDI messages, producing music from MIDI instructions, applying audio feature extraction techniques on produced music, training a machine learning classifier from these features and finally classify new instances.

4.1 Converting executable bytes to MIDI messages

Our proposed method commences with transforming byte of executable files to musical notes and generate audio files. There is an obstacle during converting each byte of executable to a specific MIDI note message, that is each byte has 256 distinct value and MIDI notes are in the range 0 to 127. According to the fact that, piano has uttermost number of pitches (notes) which is 88 and other musical instruments have less than 88 pitches, so that the number of notes even less than 128 in reality. Hence, mapping 256 state to 128 state makes some bytes to convert to the same note and causes to reduce the accuracy of our method. Our method solves this difficulty by encoding each byte to two or more notes that are played at the same time with the same duration, but on the different MIDI channels.

The byte-to-midi mapper, partitions the bits of an executable byte to different parts. In theory, there can be $2^8 - 1 = 255$ different partitioning strategies, but many strategies cannot be applied due to MIDI pitches limitation. On the other hand, some strategies provide a great number of pitches with same note but in different octave, which in this case, the overlapping of these kinds of pitches causes the decrement in the accuracy of extracted audio features. In our approach, by the aiming of remaining the process plain, the other properties of MIDI channel (for example tension or loudness of notes)

will be considered identical. For simplicity, all the channels' instrument is considered as piano, which is a standard instrument with the largest range of notes.

In the following sections, five strategies that can cover many situations and can be considered as representative of all conditions, will be investigated.

4.1.1 Strategy "4-4"

In this strategy, each byte is partitioned into two equal parts. The four significant bits are placed into the first half and the four least striking bits are located into the second half. Each half has 16 different states so it produces some pitches with alike note but different octave for diverse value of bytes.

4.1.2 Strategy "2-6"

Division size is the only difference between this strategy and the previous strategy. It partitions a byte into two segments where the first segment consists of two notable bits and second one includes other bits. This strategy still suffers from the problem of overlapping on the notes.

4.1.3 Strategy "6-2"

The strategy "6-2" is analogous to the strategy "2-6". The main difference is that the upper segment has the most remarkable six bits and the rest bits are in the lower segment.

4.1.4 Strategy "2-2-2-2"

It contains four segments and each segment comprise of two consecutive bits.

4.1.5 Strategy "1-1-1-1-1-1-1-1"

"1-1-1-1-1-1-1-1" is the simplest strategy among others. In this strategy, each bit is mapped to a peculiar note if its value is 1 also, no mapping is done in the case of 0 as its value. Therefore, we need only eight different notes and any overlapping on the note in different octaves will not be happen. Hence, the emphasis will be more on this strategy which will logically depict the best precision.

4.2 Composing the music

After preparation of MIDI file, the production of real audio signal from MIDI messages begins. This procedure requires an audio synthesizer. FluidSynth 2.0 [19] is used as our MIDI synthesizer which can generate sounds in real-time or save them in "wav" format. Music synthesizing is the most time, CPU and memory-consuming step in our method. The higher the number of MIDI notes, the costlier and time-consumer signal generation and feature extraction. Our purpose is to preserve the process fast and light, so we preferred to use only a part of the binary file to convert into audio signal such as PE header or the first 2048 byte of the file after its header or even a 1024-byte subset with a random start point. It is noticeable that using the lower quality (bitrate and sampling rate) for creating audio also is a substantial parameter to decrease the consumption of time and memory in this step. Increasing the sampling rate results better quality audio features, but also enhances the time it takes to work and the amount of resources it demands. Fortunately, the simplicity of the produced music in our method makes the sensitivity of the extracted sound features less than the bitrate and sampling rate, and the number of audio output channels. Regarding to this, we use a relatively low rate for sampling and a single channel (mono) for audio output. Consequently, the lowest possible amount is selected for sampling rate that does not affect the final accuracy remarkably. According to our experiments, this value is set to 11025 Hz.

4.3 Audio feature extraction

Our training set is constructed by audio files which are generated from executable binaries. By playing each audio, a music will be played in every tempo which contain one or more notes from one or more musical instruments. Duration of our music is computed by multiplying the number of bytes by the duration between successive beats.

MFCCs and chromagram are computed frame-by-frame that each frame is typically 10ms or 15ms, hence these features represent description of each signal frame separately. Due to this, a huge number of feature vectors will be computed that maintain and process them become tough, so these features are "down-sampled" by using mean, standard deviation or other derivative values. The features are measured from a single frame called "short-term" features. Down-sampling can be done for the whole signal or in a short time period, for example every one second which is called "mid-term features". By employing down-sampling, we lose temporal behavior of all short-term features which represents order of bytes in the original executable file. Therefore, ignoring it causes to diminish the accuracy of the classification.

As we mentioned before, we set the duration of each MIDI note to one beat which is determined by the tempo value. In fact, if the frame size is set to the duration of a beat, each feature vector keeps information about a single byte of the original file. In other cases, if the frame size is larger than a beat, we can keep information about a sequence of bytes in a single feature vector, so finding similar patterns in files will be easier than the first case. Owing to the fact, as MFCCs and chromagram are achieved from the frequency components of signal, regardless the time ordering of signal, it seems that the order of bytes (notes) in a frame does not have effect on

the values of features. As one of the metamorphic techniques is transposition of independent instructions in a small block of code, insensitivity to order of bytes in a single frame is an admirable property of using audio processing techniques for n-gram feature extraction. However, ignoring presence the bytes order does not lead to produce accurate features. To overcome this issue, overlapping in consecutive frames is employed in our approach. In this condition, a number of new frames will be created which contain information about adjacent bytes in successive frames which causes the growth in accuracy of final feature vector values.

We use the "pyAudioAnalysis" [20] library for music feature extraction. It is an open-source python library for music segmentation, classification, summarization (thumbnailing) and visualization.

4.4 Creating the model

After extracting features from the training dataset files which contain benign and malware files, we train a machine learning classifier which in our case is KNN. Although we examine more powerful classifiers, we choose KNN to illustrate that our method can gain reasonable accuracy even by employing simple classifiers.

4.5 Predicting new instances

To classify a new file, we should apply all described steps to obtain its features. Then we deliver these features to the trained classifier in order to predict its label. In malware detection, we will have two classes, malware and benign. In malware family classification, the number of classes are equal to the number of malware families.

The overall procedure of our method is demonstrated in Figure 1.

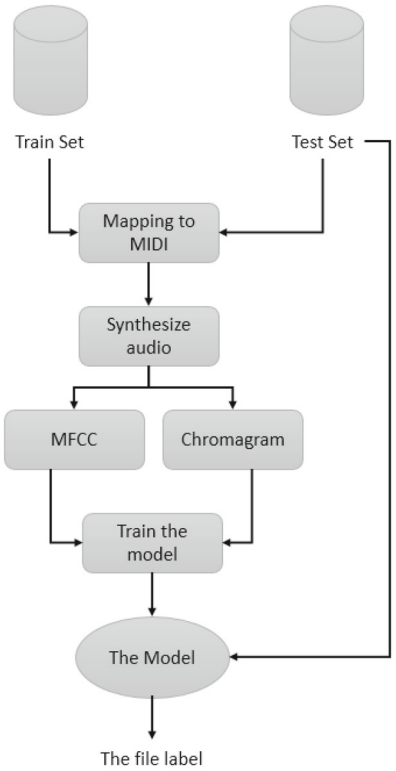

Fig. 1 The proposed method

5 Experiments and results

5.1 Datasets

Our experiments are accomplished on three datasets. Dataset 1 has 3200 samples which includes 1600 malware and 1600 benign files. These files are mixture of packed and non-packed and metamorphic files. Malware samples are randomly selected from well-known "VXHeaven" dataset in various kinds of malware. The benign dataset files are chosen from the original Windows files and other utility software. To be insure that these files are not malicious, we scanned them with ESET NOD32 and Kaspersky anti-viruses. Dataset 2 is Kaggle Microsoft Malware Classification challenge (BIG 2015) dataset which consists of nine different malware families [23]. We selected 10868 files form this dataset from all families in order to show that our approach is also effectual to classify malware based on their families.

5.2 The experimental environment

We did all experiments on the machine with following specification: Hardware: Intel Core i7-930@2.6GHz with 8 threads, 6.0GB of RAM, 7200RPM HDD. Software: Ubuntu 15.10, Python 2.6. Also, we have used sklearn machine-learning python package [22] and WEKA [21] for classification and pyAudioAnalysis [20] for MIR tasks.

5.3 Creating the model

To demonstrate our method quality, we use K-Nearest Neighbors (KNN) algorithm as a classifier [18] that is a simple classification algorithm. We tried KNN with N from 1 to 10. Moreover, we tested our method on an ensemble learning classification algorithm, Random Forest and on a Boosting algorithm, AdaBoost [18].

5.4 Cross validation

Cross Validation is a model validation technique which has the goal of solving some challenges in the classification field, such as over-fitting [25]. One of the pragmatic cross valida-

tion methods is K-fold cross validation [25]. In the K-fold method, the dataset is divided into K subsets. Each time, one of the K subsets is selected as the test set and the other K-1 folds are used as the train set. It repeats K times and then the average accuracy of all iterations considered as the final accuracy of the model. In this paper, we use 10-fold cross validation for validating the models.

5.5 Evaluation metrics

To evaluate the results, we use common machine-learning evaluation metrics. These are True Positive Ratio (TPR), False Positive Ratio (FPR), Precision, Recall, Accuracy and F-measure (F1) [24]. We will provide a brief definition of these metric in the following section.

True Positive Ratio (TPR) or Recall: True Positive Ratio (or Recall) for a certain class, is the number of samples in the class that are correctly predicted, divided by total number of samples in the class [24]. The TPR formula is defined as follow:

$$TPR\,(Recall) = \frac{TP}{TP + FN} \tag{3}$$

False Positive Ratio (FPR): False Positive Ratio for a certain class, is the number of samples in other classes that are incorrectly predicted as intended class, divided by total number of samples in the other classes [24]. The FPR formula is defined as follow:

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

Precision: Precision for a certain class is the number of samples in the class that are correctly predicted, divided by total number of samples that are predicted as the classes [24]. The precision formula is defined as follow:

$$Recall = \frac{TP}{TP + FP} \tag{5}$$

Accuracy: Accuracy is the total number of samples that are correctly predicted, divided by total number of samples [24]. The Accuracy formula is defined as follow:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6}$$

F-measure (F1): F-measure is the harmonic mean of Precision and Recall [24]. It can be applied as a common classifier performance metric. The F-measure formula is defined as below:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{7}$$

5.6 Experiments

5.6.1 Experiment on different parts of the file

Owing to the fact that, by considering all content of a file, size of the generated audio file will be huge and processing will take time and memory, we have to use only a part of each file. So, to find out which part suits out method, an experiment has been accomplished to examine the entire PE header, the strings inside the file, instruction opcodes and a piece of files' code section which has the goal of investigating the effect of each part on the final result of our approach. According to the number of distinguished opcode which is more than 256, our mapper employs nine channels instead of eight channels for mapping each opcode to a MIDI. The block size for string and code section is 4096 bytes while for opcode the first 4096 opcodes are considered. Results of examinations are illustrated in Table 1 and Table 2. Also, the F-measure value for different parts of the file is depicted in Figure 2.

Regarding the result, by considering only the header of the file, the accuracy of the proposed method is reasonably better than other methods. It is not astonishing, because, header is a short-sized piece of informative and is the only part of the file that is not encrypted. It is true that the opcodes give us a refined and more accurate information, but because of the limited ability to capture all of them in our method, the accuracy is reduced slightly. Also, our goal is to demonstrate that malware detection is prone to increasing the speed and simplicity without disassembling, which is a time-consuming process and, in cases where anti-disassembling techniques are used, is prone to error detection.

Due to the fact that each instruction code will contain opcode and operands, an instruction code has more bytes in comparison to an opcode, and the number of instructions that are used to generate fixed size music (we consider 4096 notes per music) is fewer than that for opcodes to be attended alone, therefore less data are provided in this case.

Also, the file string has a lower result than the rest of the state. File strings are not always inside the files as hard-coded. Moreover, regarding to our method, the processing of each frame is accomplished in the frequency domain, the ordering of strings is not considered within a frame which is important for strings unlike adjacent instruction codes and opcodes and leads to reduction in accuracy.

Due to the mentioned reasons, PE header is considered as the input of our next experiments.

5.6.2 The Experimental results on different mapping strategies

To complete our experiment, among various mapping strategies explained in previous sections, five strategies that cover all different possible situations are selected. This experiment

Table 1 Comparing TPR and FPR for different parts of file as input of each sample

Classifier	True Positive Rate (TPR)				False Positive Rate (FPR)			
	Header	Code	Opcode	Strings	Header	Code	Opcode	Strings
KNN(N = 1)	93.60	87.42	90.40	75.05	7.60	16.16	14.40	17.01
KNN(N = 2)	93.60	87.42	90.40	75.05	7.60	16.16	13.80	17.21
KNN(N = 3)	94.60	88.64	90.60	77.89	8.60	13.94	13.80	17.83
KNN(N = 4)	94.20	89.05	93.20	77.48	7.20	14.55	11.00	16.60
KNN(N = 5)	94.60	90.87	91.80	76.67	8.00	15.96	12.40	17.21
KNN(N = 6)	94.40	90.67	92.40	77.48	8.00	15.15	12.00	16.19
KNN(N = 7)	95.00	90.06	92.80	76.27	7.60	15.56	11.40	16.60
KNN(N = 8)	94.80	91.08	92.80	77.08	7.40	15.15	11.00	15.98
KNN(N = 9)	94.80	90.87	91.60	75.86	7.60	15.56	13.00	17.21
KNN(N = 10)	94.60	90.87	93.20	76.47	7.60	15.35	10.80	17.21
SVM	94.80	87.63	93.00	81.74	5.20	19.39	8.20	18.03
AdaBoost	**95.60**	92.70	93.00	86.82	3.80	12.53	7.60	18.85
Random Forest	**95.60**	92.70	93.40	86.82	**3.40**	12.53	6.20	18.24

Table 2 Comparing results for different parts of file as input of each sample

Classifier	Mean Accuracy				Mean F-Measure			
	Header	Code	Opcode	Strings	Header	Code	Opcode	Strings
KNN(N = 1)	93.00	85.63	88.28	79.00	93.04	85.86	88.00	78.22
KNN(N = 2)	93.00	85.63	88.54	78.90	93.04	85.86	88.30	78.14
KNN(N = 3)	93.00	87.35	88.64	80.02	93.11	87.49	88.40	79.67
KNN(N = 4)	93.50	87.25	91.28	80.43	93.55	87.45	91.10	79.92
KNN(N = 5)	93.30	87.45	89.91	79.71	93.39	87.84	89.70	79.16
KNN(N = 6)	93.20	87.75	90.41	80.63	93.28	88.08	90.20	80.08
KNN(N = 7)	93.70	87.25	90.89	79.82	93.78	87.57	90.70	79.16
KNN(N = 8)	93.70	87.96	91.06	80.53	93.77	88.30	90.90	79.92
KNN(N = 9)	93.60	87.65	89.54	79.31	93.68	88.02	89.30	78.65
KNN(N = 10)	93.50	87.75	91.37	79.61	93.57	88.10	91.20	79.04
SVM	94.80	84.11	92.44	81.86	94.80	84.62	92.40	81.91
AdaBoost	95.90	90.08	92.72	84.00	95.89	90.32	92.70	84.50
Random Forest	**96.10**	90.08	93.58	84.30	**96.08**	90.32	93.60	84.75

is accomplished on Dataset 1 by considering the PE header of each file as the input.

Results of the experiment are listed in Table 3 and Table 4. Also, the F-measure values for different mapping strategies are illustrated in Figure 3.

It is clear from the achieved results that through mapping strategies, the result of strategy "1-1-1-1-1-1-1-1" is outstanding. To describe this, it must be mentioned that, musical pitches have a variety of frequency and different musical instruments have diverse note shapes. Octave and note are momentous properties of a pitch. However, MFCC is frequency sensitive, chromagram is based on notes and doesn't susceptible to octaves. Regarding our investigation on all notes, the selected instrument (which have different note shapes) for producing the final audio cannot affect the MFCC or chromagram results severely. In fact, a strategy could achieve the admissible accuracy if is capable of mapping the different binary bytes to pitches where for any of two different bytes do not produce two identical notes even in various octaves. Therefore, in strategies with more than twelve mapping states, same pitches with same notes but different octaves are employed for mapping various bytes which make the effectiveness of chromagram features less. In the "1-1-1-1-1-1-1-1" strategy, each bit in a byte is mapped to a diverse pitch, hence eight variate pitches are demanded for mapping a byte to a MIDI message uttermost which takes the benefit of non-overlapping in the notes with different octaves. Moreover, for making the strategy mightier, we put the eight various pitches in disparate octaves that makes them more frequency separable.

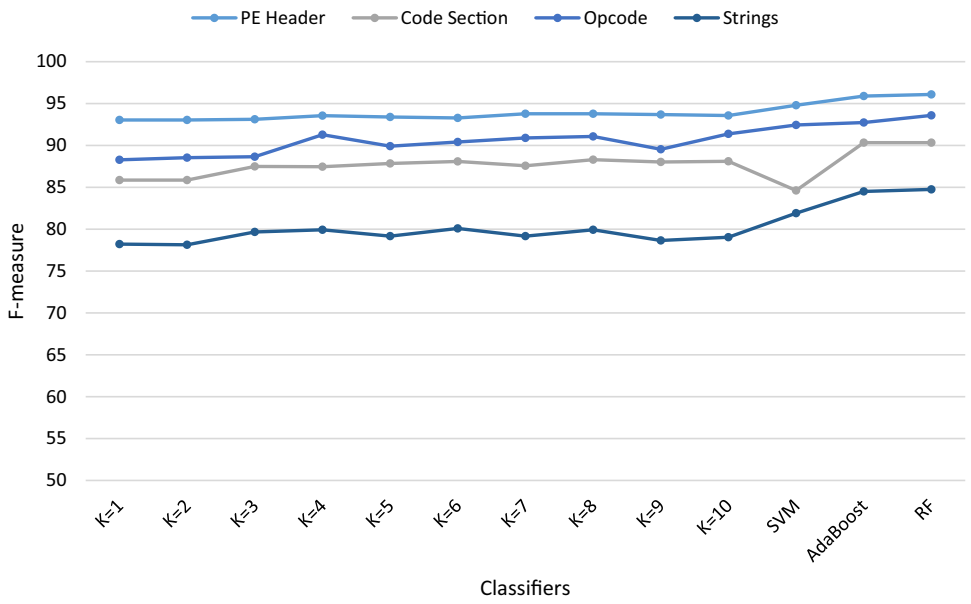

Fig. 2 F-measure values for experiment on different parts of file

Table 3 Comparing TPR and FPR for different mapping strategies

Classifier	True Positive Rate (TPR)					False Positive Rate (FPR)				
	Str 1[a]	Str 2[b]	Str 3[c]	Str 4[d]	Str 5[e]	Str 1	Str 2	Str 3	Str 4	Str 5
KNN(N = 1)	88.00	91.80	91.20	91.40	93.60	12.20	12.20	12.60	12.60	7.60
KNN(N = 2)	88.00	91.80	91.20	91.40	93.60	12.20	12.20	12.60	12.60	7.60
KNN(N = 3)	91.80	92.20	93.40	93.20	94.60	9.80	11.60	13.40	12.40	8.60
KNN(N = 4)	91.60	93.00	93.80	93.00	94.20	9.40	12.20	13.00	11.40	7.20
KNN(N = 5)	92.20	93.00	93.60	93.80	94.60	9.60	13.00	15.40	11.40	8.00
KNN(N = 6)	92.00	93.00	94.20	93.80	94.40	9.80	12.20	14.20	11.80	8.00
KNN(N = 7)	91.40	93.80	94.00	94.60	95.00	11.80	13.40	14.60	12.40	7.60
KNN(N = 8)	91.20	93.60	94.40	94.40	94.80	11.40	13.20	14.60	11.40	7.40
KNN(N = 9)	90.80	92.80	94.80	94.60	94.80	11.20	11.40	14.40	12.60	7.60
KNN(N = 10)	91.40	92.80	94.80	94.00	94.60	11.00	12.00	14.00	11.60	7.60
SVM	90.80	91.40	93.40	93.80	94.80	10.40	13.40	17.60	11.00	5.20
AdaBoost	94.80	**96.20**	95.40	93.80	95.60	7.40	10.80	12.60	8.00	3.80
Random Forest	94.60	96.00	95.80	94.00	95.60	7.40	10.40	11.80	7.60	**3.40**

[a]Strategy "2-6"
[b]Strategy "6-2"
[c]Strategy "4-4"
[d]Strategy "2-2-2-2"
[e]Strategy "1-1-1-1-1-1-1-1"

Further experiments have been done by using the best strategy (Strategy "1-1-1-1-1-1-1-1") achieved in this step.

5.6.3 Experiment for considering audio features

In this examination, the impact of different audio features will be investigated in three various situations as follows:

- Only MFCC is extracted from audio signals.
- Only chromagram is exploited.
- Both features are extracted and considered.

The result of our experiment is illustrated in Table 5 and Table 6. The F-measure values for different parts of the file are depicted in Figure 4.

In music classification, in order to capture both the rhythmic and emotional state of a signal and its harmonic, both features are used commonly. Due to the fact that MFCC is a signal perception feature, it represents more appropriate components for describing signals when the only MFCC is employed. In our method, owing to the plain structure of our produced music where there is no singer or other sound effect and the notes are equally distributed and stretched, the distri-

Table 4 Comparing results for different mapping strategies

Classifier	Accuracy					F-measure				
	Str 1[a]	Str 2[b]	Str 3[c]	Str 4[d]	Str 5[e]	Str 1	Str 2	Str 3	Str 4	Str 5
KNN(N = 1)	87.90	89.80	89.30	89.40	93.00	87.91	90.00	89.50	89.61	93.04
KNN(N = 2)	87.90	89.80	89.30	89.40	93.00	87.91	90.00	89.50	89.61	93.04
KNN(N = 3)	91.00	90.30	90.00	90.40	93.00	91.07	90.48	90.33	90.66	93.11
KNN(N = 4)	91.10	90.40	90.40	90.80	93.50	91.14	90.64	90.72	91.00	93.55
KNN(N = 5)	91.30	90.00	89.10	91.20	93.30	91.38	90.29	89.57	91.42	93.39
KNN(N = 6)	91.10	90.40	90.00	91.00	93.20	91.18	90.64	90.40	91.25	93.28
KNN(N = 7)	89.80	90.20	89.70	91.10	93.70	89.96	90.54	90.12	91.40	93.78
KNN(N = 8)	89.90	90.20	89.90	91.50	93.70	90.03	90.52	90.33	91.74	93.77
KNN(N = 9)	89.80	90.70	90.20	91.00	93.60	89.90	90.89	90.63	91.31	93.68
KNN(N = 10)	90.20	90.40	90.40	91.20	93.50	90.32	90.62	90.80	91.44	93.57
SVM	90.20	89.00	87.90	91.40	94.80	90.26	89.26	88.53	91.60	94.80
AdaBoost	93.70	92.70	91.40	92.90	95.90	93.77	92.95	91.73	92.96	95.89
Random Forest	93.60	92.80	92.00	93.20	**96.10**	93.66	93.02	92.29	93.25	**96.08**

[a]Strategy "2-6"
[b]Strategy "6-2"
[c]Strategy "4-4"
[d]Strategy "2-2-2-2"
[e]Strategy "1-1-1-1-1-1-1-1"

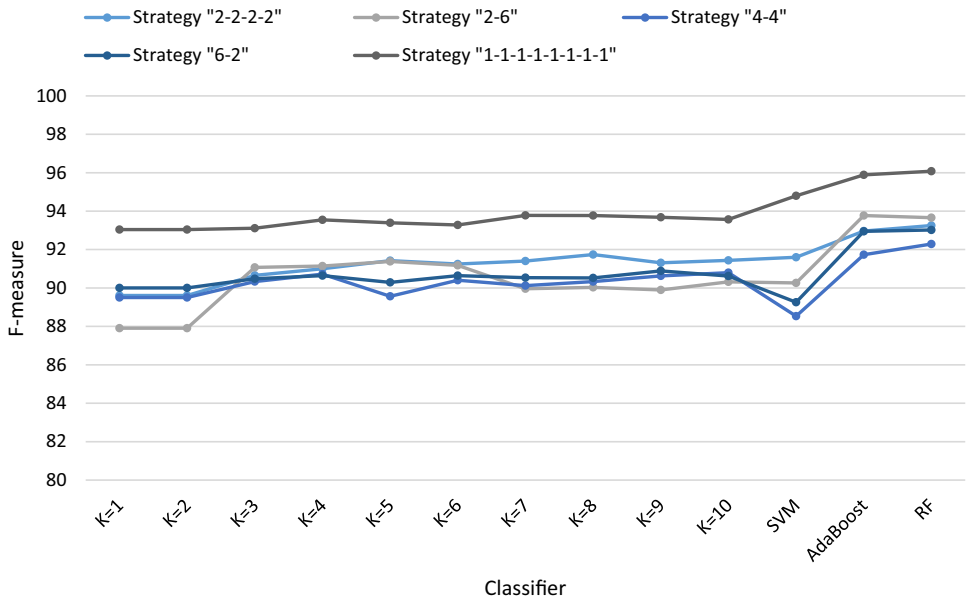

Fig. 3 F-measure values for experiment on different mapping strategies

bution of the note described by chromagram produce similar results to MFCC. That's why there is no significant difference between the selection of only one feature or both. Indeed, in other strategies which have octave overlap, due to not taking into account octaves in the chromagram, the result of MFCC is more reasonable than chromagram. Regarding the results, using both of the features has slightly better results on average compared to applying only one feature. Because of the no remarkable difference in the extraction of both features compared to the time when only one feature is extracted, we will continue the experiments by exploiting both attributes.

5.6.4 Experiments on comparison with other byte-level methods

In this section, we compare the performance of our method, which is a frequency-based approach, with Khorsand [10] and Kolter [12] approaches that work directly on the byte sequence. This test is done on Dataset 1 and for all methods, we will only consider the PE header of each executable file. Results of the experiment is illustrated in Table 7.

In fact, comparing to other competetition methods, the accuracy of our approach is better on this dataset while it

Table 5 Comparing TPR and FPR obtained by different audio features

Classifier	True Positive Rate (TPR)			False Positive Rate (FPR)		
	MFCC	Chromagram	Both	MFCC	Chromagram	Both
KNN(N = 1)	95.00	93.00	93.60	8.20	10.00	7.60
KNN(N = 2)	95.00	93.00	93.60	8.20	10.00	7.60
KNN(N = 3)	94.40	95.20	94.60	7.20	8.80	8.60
KNN(N = 4)	94.20	94.80	94.20	7.60	8.40	7.20
KNN(N = 5)	94.80	95.60	94.60	8.20	8.20	8.00
KNN(N = 6)	95.20	95.60	94.40	8.60	8.40	8.00
KNN(N = 7)	94.60	95.40	95.00	9.20	8.20	7.60
KNN(N = 8)	95.00	95.60	94.80	9.00	8.80	7.40
KNN(N = 9)	94.60	95.40	94.80	9.60	9.40	7.60
KNN(N = 10)	94.80	95.20	94.60	9.80	8.20	7.60
SVM	94.80	94.80	94.80	5.00	5.40	5.20
AdaBoost	95.20	**95.60**	**95.60**	**3.40**	5.80	3.80
Random Forest	95.20	**95.60**	**95.60**	4.00	5.00	**3.40**

Table 6 Comparing results obtained by different audio features

Classifier	Mean Accuracy			Mean F-Measure		
	MFCC	Chromagram	Both	MFCC	Chromagram	Both
KNN(N = 1)	93.40	91.50	93.00	93.50	91.63	93.04
KNN(N = 2)	93.40	91.50	93.00	93.50	91.63	93.04
KNN(N = 3)	93.60	93.20	93.00	93.65	93.33	93.11
KNN(N = 4)	93.30	93.20	93.50	93.36	93.31	93.55
KNN(N = 5)	93.30	93.70	93.30	93.40	93.82	93.39
KNN(N = 6)	93.30	93.60	93.20	93.42	93.73	93.28
KNN(N = 7)	92.70	93.60	93.70	92.84	93.71	93.78
KNN(N = 8)	93.00	93.40	93.70	93.14	93.54	93.77
KNN(N = 9)	92.50	93.00	93.60	92.65	93.16	93.68
KNN(N = 10)	92.50	93.50	93.50	92.67	93.61	93.57
SVM	94.90	94.70	94.80	94.89	94.71	94.80
AdaBoost	95.90	94.90	95.90	95.87	94.94	95.89
Random Forest	95.60	95.30	**96.10**	95.58	95.31	96.08

cannot be argued that our method is always better than other methods that are work directly without mapping the file to the frequency domain. Indeed, our goal is to depict that by converting the executable files to audio signals, algorithms in the field of audio can be used for feature extraction which tackle the problem of finding appropriate feature set in other heuristic malware detection methods and increase the accuracy sensibly.

5.6.5 Experimental results for malware family classification

To demonstrate the potency of our work in the field of malware family classification, we assess the method on Kaggle Microsoft Malware Classification Challenge (BIG 2015) dataset with 10868 samples selected from nine different families with variate size. The results of experiment are listed in the Table 8. Also, the F-measure values for different classifiers are illustrated in Figure 5.

From the results, it is obvious that our method is able to obtain reasonable results on huge and unbalanced dataset. In music genre classification which is a multi-class classification, the nature of issue is to classify music into various genres. Thus, our technique can also be used for detecting family of unknown malware based on seen families in the test dataset which shows the ability of our method to classify malware based on their family.

5.6.6 Comparison with other methods

Opcodes are more transparent and potent so to evaluate the performance of our method, we compare it with two powerful Hashemi [16] and Santos [5] methods which are based

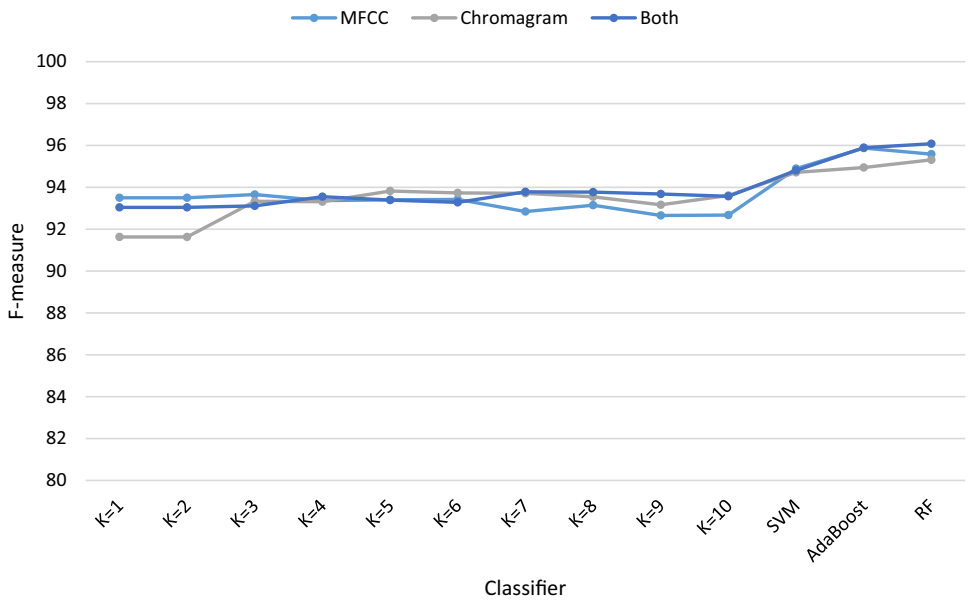

Fig. 4 F-measure values for experiment on different audio features

Table 7 Results of comparison with other byte-level methods

Classifier	True Positive Rate (TPR)	False Positive Rate (FPR)	Accuracy	F-Measure
Proposed Method	**95.60**	**3.40**	**96.10**	**96.08**
Khorsand [10]	96.00	9.00	93.60	93.78
Kolter [12]	92.00	11.00	90.20	90.37

Table 8 Comparing results obtained from malware family classification dataset

Classifier	True Positive Rate (TPR)	False Positive Rate (FPR)	Accuracy	F-Measure
KNN(N = 1)	94.70	9.00	94.73	94.60
KNN(N = 2)	94.70	9.00	94.73	94.60
KNN(N = 3)	94.50	9.00	94.46	94.30
KNN(N = 4)	94.80	9.00	94.77	94.60
KNN(N = 5)	94.40	10.00	94.37	94.20
KNN(N = 6)	94.30	10.00	94.25	94.10
KNN(N = 7)	94.00	11.00	94.00	93.80
KNN(N = 8)	93.90	11.00	93.93	93.70
KNN(N = 9)	93.80	12.00	93.76	93.60
KNN(N = 10)	93.60	12.00	93.61	93.40
SVM	94.20	12.00	94.21	94.10
AdaBoost	95.70	9.00	95.65	95.60
Random Forest	**95.90**	**8.00**	**95.87**	**95.80**

on opcodes. Due to the significant evolution of heuristic opcode-based approaches, the practicality of our method is demonstrated when it compared to the two mentioned techniques. This experiment is performed on Dataset 1. In this experiment, we run the methods on the same environment which is explained in Section 5.2. According to the necessity of opcodes in Hashemi [16] and Santos [5] methods, a disassembler is demanded to extract the opcodes of files. We choose IDA Pro tools in our work for extracting opcodes.

The experiment results are listed in Table 9 and Table 10. Figure 6 illustrate the comparison of F-measure on the different methods. The running time and memory usage from the preprocessing of file to achieving the results is listed in Table 11.

Regarding the results, our method achieves better performance. This might be because of the presence of anti-disassembly techniques in some malware files which in this case a huge volume of opcodes will not be extracted and input

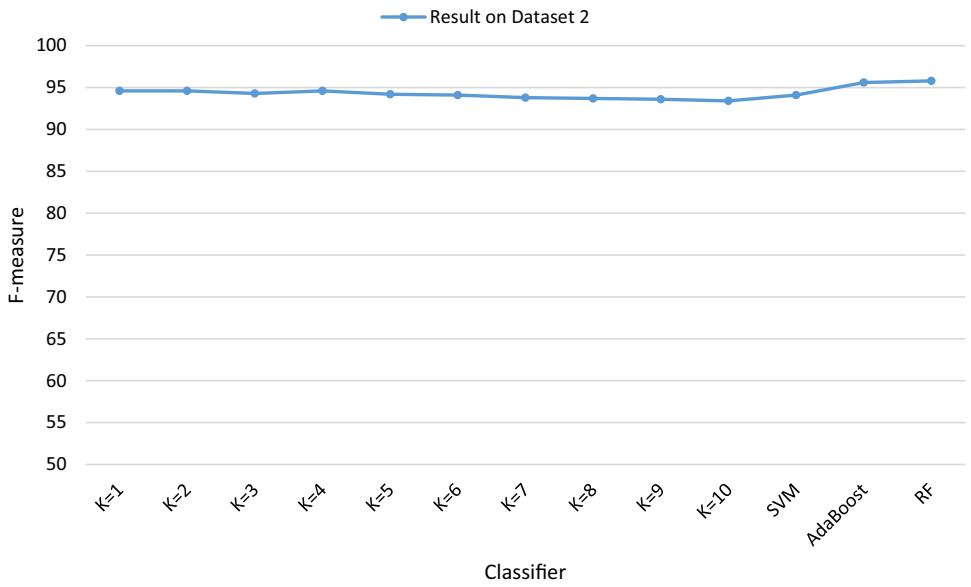

Fig. 5 F-measure values for experiment on Dataset 2

Table 9 TPR and FPR results of comparison with other methods

Classifier	True Positive Rate (TPR)			False Positive Rate (FPR)		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	93.60	90.38	92.79	7.60	12.60	13.60
KNN(N = 2)	93.60	90.78	92.79	7.60	13.00	15.80
KNN(N = 3)	94.60	92.38	92.79	8.60	14.20	18.40
KNN(N = 4)	94.20	92.18	93.19	7.20	14.40	18.40
KNN(N = 5)	94.60	93.39	93.59	8.00	13.60	21.60
KNN(N = 6)	94.40	93.99	93.79	8.00	14.00	23.60
KNN(N = 7)	95.00	94.39	93.99	7.60	15.00	26.60
KNN(N = 8)	94.80	94.59	93.79	7.40	15.60	26.20
KNN(N = 9)	94.80	94.79	94.79	7.60	16.20	28.40
KNN(N = 10)	94.60	95.19	94.99	7.60	17.00	29.20
SVM	94.80	**96.59**	94.59	5.20	11.00	9.20
AdaBoost	95.60	94.19	95.39	3.80	12.00	8.60
Random Forest	95.60	94.99	95.39	**3.40**	11.80	8.00

of the opcode-based methods will be imperfect. Owing to the fact that our approach employ bytes directly as inputs, these features are always available. Also, our technique considers PE header as input which is not encrypted and contains useful information.

In byte-level methods, each byte can be used as a part of the input and there is no limit in our approach on the input range. Selecting a larger chunk of the file can improve accuracy but also increases running time. PE-header, resource section and data section bytes have beneficial information whereas in other approaches these parts may be ignored. For example, in opcode-based techniques, only opcodes of a file are applied as input and other sections are relinquished.

One of the main features in the most of audio processing techniques such as MFCC is robustness in noisy environments. It can help us for finding partial similarity between different similar-functionality samples which can affect the accuracy of method in confronting morphed samples. Hence, the results of our method has been better. In fact, this claim in not true always and we still do not contend that our method will work perfect in any condition.

In addition, our technique is quick and require less resources than other mentioned methods because it does not need disassembling and audio processing methods are fast and lightweight. Hashemi's approach [16] consumes a lot of time due to the iteration method for convergence and Santos's method [5] requires a lot of memory to keep the features.

Table 10 Results of comparison with other methods

Classifier	Mean Accuracy			Mean F-Measure		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	93.00	88.89	89.59	93.04	89.04	89.90
KNN(N = 2)	93.00	88.89	88.49	93.04	89.09	88.95
KNN(N = 3)	93.00	89.09	87.19	93.11	89.43	87.86
KNN(N = 4)	93.50	88.89	87.39	93.55	89.23	88.07
KNN(N = 5)	93.30	89.89	85.99	93.39	90.22	86.96
KNN(N = 6)	93.20	89.99	85.09	93.28	90.37	86.27
KNN(N = 7)	93.70	89.69	83.68	93.78	90.14	85.20
KNN(N = 8)	93.70	89.49	83.78	93.77	89.99	85.25
KNN(N = 9)	93.60	89.29	83.18	93.68	89.84	84.92
KNN(N = 10)	93.50	89.09	82.88	93.57	89.71	84.72
SVM	94.80	92.79	92.69	94.80	93.05	92.82
AdaBoost	95.90	91.09	93.39	95.89	91.35	93.52
Random Forest	**96.10**	91.59	93.69	**96.08**	91.86	93.79

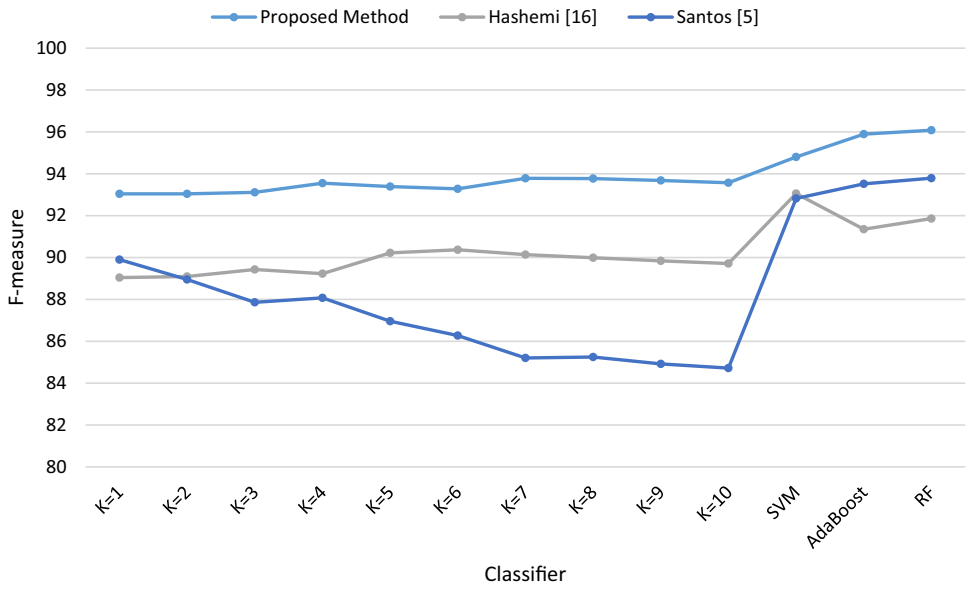

Fig. 6 Comparison of F-measure values for different methods

Table 11 Comparing time and memory consumption for different methods

Method	Running Time (seconds)	Memory (megabytes)
Proposed Method	**4850**	**145**
Hashemi [16]	349,380	21,407
Santos [5]	26,780	7,156

5.6.7 Experiments on obfuscated samples

In this section, two experiments will be conducted to investigate the accuracy of the proposed method, Hashemi [16] and Santos [5] approaches when obfuscated samples are employed. First examination is on packed samples. Therefore, all samples of Dataset 1 (either benign or malware) are packed by the most commonly used UPX packer with its default settings. The second experiment is on metamorphic samples. To do this, all the samples of Dataset 1 (either benign or malware) are morphed by the….which is an open-source metamorphic engine tool.

The experiment results for packed dataset are listed in Table 12 and Table 13. Figure 7 illustrate the comparison of F-measure on the different methods. The experiment results for obfuscated dataset are listed in Table 14 and Table 15. Figure 8 illustrate the comparison of F-measure on the different methods.

It can be observed from the result that the final accuracy of our method in considerably higher in comparison to the other

Table 12 TPR and FPR results of comparison with other methods on packed dataset

Classifier	True Positive Rate (TPR)			False Positive Rate (FPR)		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	90.20	45.47	55.40	13.10	10.59	18.40
KNN(N = 2)	90.20	37.22	54.00	13.10	4.89	19.40
KNN(N = 3)	91.00	43.06	56.40	11.51	6.92	19.00
KNN(N = 4)	91.20	38.03	56.00	11.51	**3.67**	17.80
KNN(N = 5)	91.60	40.04	55.20	11.71	5.70	18.60
KNN(N = 6)	91.40	38.03	54.80	12.10	4.89	19.20
KNN(N = 7)	91.40	39.03	54.80	11.71	5.30	19.00
KNN(N = 8)	91.60	37.63	54.80	11.90	4.89	19.80
KNN(N = 9)	91.80	39.03	54.80	12.10	5.70	20.20
KNN(N = 10)	92.00	38.03	55.80	12.10	5.70	20.20
SVM	93.40	52.52	63.20	9.13	13.44	19.00
AdaBoost	**94.00**	53.92	64.40	9.13	7.74	18.20
Random Forest	93.20	53.80	65.20	8.53	6.80	17.60

Table 13 Results of comparison with other methods on packed dataset

Classifier	Mean Accuracy			Mean F-Measure		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	88.55	67.31	68.50	88.69	58,32	63.75
KNN(N = 2)	88.55	65.99	67.30	88.69	52.41	62.28
KNN(N = 3)	89.74	67.91	68.70	89.83	57.45	64.31
KNN(N = 4)	89.84	67.00	69.10	89.94	53.69	64.44
KNN(N = 5)	89.94	67.00	68.30	90.07	54.97	63.52
KNN(N = 6)	89.64	66.40	67.80	89.78	53.24	62.98
KNN(N = 7)	89.84	66.70	67.90	89.96	54.11	63.01
KNN(N = 8)	89.84	69.19	67.50	89.98	52.82	62.77
KNN(N = 9)	89.84	66.50	67.30	90.00	53.96	62.62
KNN(N = 10)	89.94	66.99	67.80	90.11	52.94	63.40
SVM	92.13	69.43	72.10	92.20	63.35	69.37
AdaBoost	**92.43**	72.98	73.10	**92.52**	66.75	70.53
Random Forest	92.33	73.50	73.80	92.37	66.99	71.33

techniques. Generally, UPX encrypts the data and code section and locate them to a new section called "UPX 0". Also, it leaves the header (other than the Import Address Table) and the resource section intact. Then, it puts the unpacker's code in a new section called "UPX 1" where the entry point address of program is also placed. In this case, our method which utilizes the PE header as the input of each instance, will still have useful information while Hashemi [16] and Santos [5] methods need to be disassemble the samples. Hence, unpacker's code is the only part of the codes which is not encrypted and from the survey, it is clear that most of the extracted code for the instances will be identical. Owing to the difference in the size and the number of sections in samples, some of the unpack codes have trivial discrepancy with each other that increases the accuracy of the opcode-based methods more than 50%.

Of course, by referring to these results, it cannot be decided that our proposed method will always respond well to encountered packed sample. The precision of our method will not be promising when the high percentage of files are packed and the fundamental changes are made to the structure of the packed files (for example, using commercial packers or customized UPX).

In the second experiment, the mentioned metamorphic engine does not change much in other sections, such as header, because it only alters the structure of the code. In this situation, our method will be more acceptable. Of course, when using stronger morphing engines, the whole structure

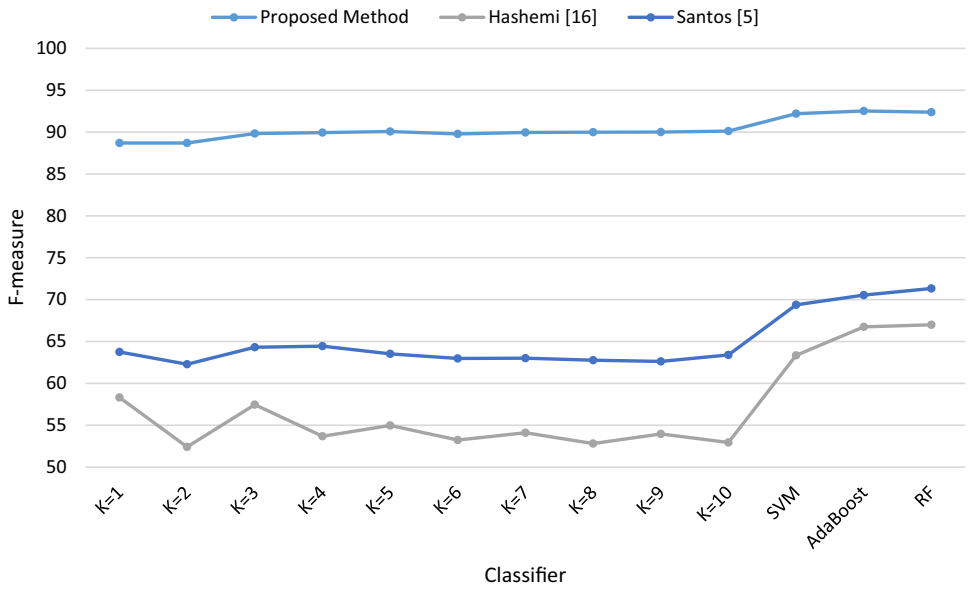

Fig. 7 Comparison of F-measure values for different methods on packed dataset

Table 14 TPR and FPR results of comparison with other methods on obfuscated dataset

Classifier	True Positive Rate (TPR)			False Positive Rate (FPR)		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	87.60	71.60	72.80	9.90	25.20	25.90
KNN(N = 2)	87.60	72.40	73.20	10.30	24.20	25.90
KNN(N = 3)	89.50	77.70	74.80	9.10	21.20	22.60
KNN(N = 4)	89.50	77.70	75.70	9.10	21.00	22.60
KNN(N = 5)	91.30	73.40	77.10	8.50	23.00	22.60
KNN(N = 6)	92.30	73.40	75.90	8.10	23.20	24.40
KNN(N = 7)	92.30	71.40	76.30	7.10	23.20	23.60
KNN(N = 8)	91.50	71.40	81.10	7.90	25.70	19.80
KNN(N = 9)	91.50	71.00	81.30	8.10	26.30	18.40
KNN(N = 10)	90.90	70.20	81.90	8.70	26.70	18.40
SVM	93.70	76.90	81.50	**5.70**	18.60	18.00
AdaBoost	94.50	79.30	81.90	**5.70**	18.60	18.00
Random Forest	**94.70**	80.70	83.20	5.30	18.20	16.00

of the file may be modified, which in this case our mode will not be promising either.

Ultimately, we do not claim that our method works better than opcode-based methods against obfuscating techniques. However, it can be seen that there are some examples of obfuscated files in the real world that can be more accurately detected due to the fact that all parts of the file are not obfuscated.

6 Summary and conclusion

In this paper, we propose a novel method for detecting malicious files by employing music classification techniques in order to classify unknown files. We define some mapping strategy to converting file bytes to MIDI messages. Then, we synthesis MIDI files to produce an audio signal. Several commonly-used audio features are extracted from the audio in order to train the machine-learning classifiers.

Due to the byte-level essence of our method, it does not require any disassembling technique which is error prone in several modern malware created with obfuscation techniques. In addition, our approach does not contain any highly influential parameter which makes parameter selection an easy task. Also, in our method, any byte of file can be selected as input such as header, code section, data section and resource section. Although opcodes represent more pleasant information of files which causes the higher clas-

Table 15 Results of comparison with other methods on obfuscated dataset

Classifier	Mean Accuracy			Mean F-Measure		
	Proposed Method	Hashemi [16]	Santos [5]	Proposed Method	Hashemi [16]	Santos [5]
KNN(N = 1)	87.80	72.20	72.60	88.70	72.63	73.26
KNN(N = 2)	87.60	73.20	72.80	88.52	73.60	73.52
KNN(N = 3)	89.10	77.30	75.20	90.09	78.08	75.77
KNN(N = 4)	89.10	77.40	75.60	90.09	78.16	76.27
KNN(N = 5)	90.30	74.30	76.30	91.38	74.71	77.15
KNN(N = 6)	91.00	74.20	74.80	92.10	74.63	75.70
KNN(N = 7)	91.50	73.20	75.40	92.57	73.33	76.26
KNN(N = 8)	90.70	72.00	79.70	91.75	72.42	80.72
KNN(N = 9)	90.60	71.50	80.50	91.66	71.94	81.42
KNN(N = 10)	90.00	70.90	80.80	91.05	71.26	81.78
SVM	92.90	78.20	80.80	93.99	78.63	81.70
AdaBoost	93.30	79.40	81.00	94.42	80.12	81.94
Random Forest	**93.60**	80.30	82.60	**94.62**	81.14	83.50

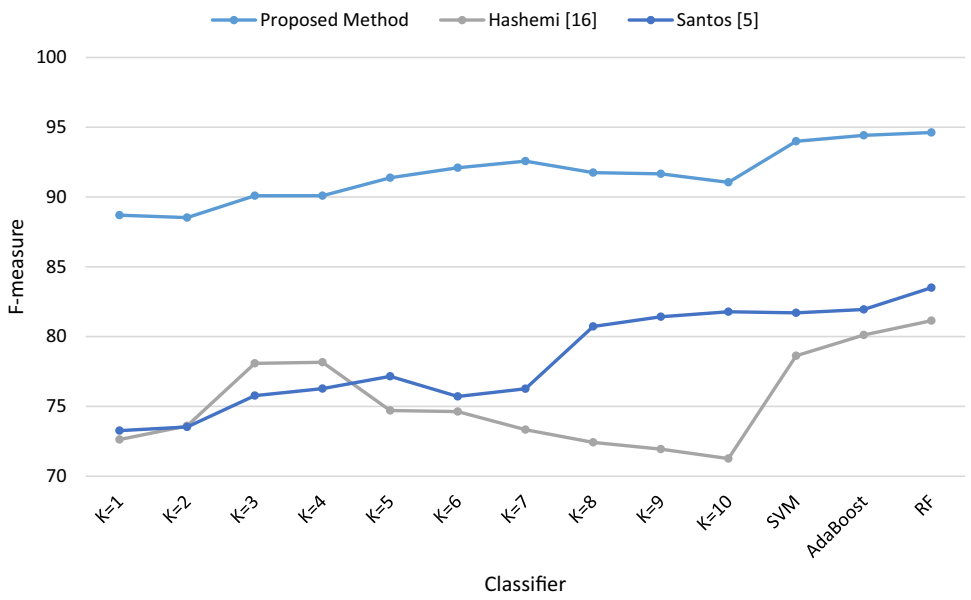

Fig. 8 Comparison of F-measure values for different methods on obfuscated dataset

sification accuracy, it needs the fallible and time-consuming preprocessing task.

In compression-based approaches, a massive memory is required for maintaining the compression models and the number and size of the train set files is limited, so these methods cannot be applied by antivirus applications in production world.

In comparison to the other signal-based malware detection techniques that attempt to convert the file to a 2-D or 3-D image, our method is based on the 1-D audio signal that is simpler to process, faster and more resource-thrifty. Furthermore, byte-level image-based techniques can be evaded easily by embedded grayscale pictures in the resource section of the file.

In fact, multitude ways are available for improving our method. First, new strategies for mapping binary files to the audio signals can be developed which can increase the final accuracy. Second, a wider range of bytes can be selected as input such as PE section along with resource section and code section. It is considerable that, by employing offline signal synthesizing we can reduce the processing time.

References

1. Moir, R.: Defining Malware: FAQ. Microsoft TechNet. https://technet.microsoft.com/en-us/library/dd632948.aspx (2003). Accessed 17 Feb 2017

2. Symantec.: Internet Security Threat Report, Volume 17. Technical report, Symantec Corporation (2011). http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf. Accessed 19 May 2018
3. Vinod, P., Jaipur, R., Laxmi, V., Gaur, M.: Survey on malware detection methods. In: Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09), pp. 74–79 (2009)
4. Wong, W.: Analysis and detection of metamorphic computer viruses. Department of Computer Science, San Jose State University, May, Master's Thesis (2006)
5. Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G.P.: Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inf. Sci. (Ny) **231**, 64–82 (2013)
6. Typke, R., Wiering, F., Veltkamp, R.C.: A survey of music information retrieval systems. In: ISMIR, pp. 153–160 (2005)
7. Fu, Z., Lu, G., Ting, K.M., Zhang, D.: A survey of audio-based music classification and annotation. IEEE Trans. Multimed. **13**(2), 303–319 (2011)
8. Tiwari, V.: MFCC and its applications in speaker recognition. Int. J. Emerg. Technol. **1**(1), 19–22 (2010)
9. Zhou, Y., Inge, W.M.: Malware detection using adaptive data compression. In: Proceedings of the 1st ACM Workshop on Workshop on AISec, pp. 53–60 (2008)
10. Khorsand, Z., Hamzeh, A.: A novel compression-based approach for malware detection using PE header. In: 2013 5th Conference on IEEE Information and Knowledge Technology (IKT), pp. 127–133 (2013)
11. Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proceedings. 2001 IEEE Symposium on Security and Privacy, 2001. S\&P 2001, pp. 38–49 (2001)
12. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**(Dec), 2721–2744 (2006)
13. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B. S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, vol. 4 (2011)
14. Han, K.S., Lim, J.H., Kang, B., Im, E.G.: Malware analysis using visualized images and entropy graphs. Int. J. Inf. Secur. **14**(1), 1–14 (2015)
15. Nataraj, L., Yegneswaran, V., Porras, P., Zhang, J.: A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 21–30 (2011)
16. Hashemi, H., Azmoodeh, A., Hamzeh, A., Hashemi, S.: Graph embedding as a new approach for unknown malware detection. J. Comput. Virol. Hacking Tech. **13**(3), 153–166 (2017)
17. Yu, X., Zhang, J., Liu, J., Wan, W., Yang, W.: An audio retrieval method based on chromagram and distance metrics. In: 2010 International Conference on. IEEE Audio Language and Image Processing (ICALIP), pp. 425–428 (2010)
18. Harrington, P.: Machine Learning in Action, no. 3, vol. 37. Manning Publications Co., Greenwich, CT, USA (2012)
19. FluidSynth 2.0. http://www.fluidsynth.org/, Accessed 17 Feb 2017
20. Giannakopoulos, T.: pyAudioAnalysis: an open-source python library for audio signal analysis. PLoS ONE **10**(12), 1–17 (2015)
21. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Vanderplas, J.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
23. Microsoft Malware Classification Challenge (BIG 2015), Kaggle. https://www.kaggle.com/c/malware-classification. Accessed 17 Feb 2017
24. Powers, D.M.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. J. Mach. Learn. Technol. **2**(1), 37–39 (2011)
25. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 1995 International Joint Conference on Artificial Intelligence, vol. 14, no. 2, pp. 1137–1145 (1995)
26. Dodge, C., Jerse, T.A.: Computer music: synthesis, composition and performance. Macmillan Library Reference, Hampshire (1997)
27. Bello, J. P.: MIDI Code, NewYork University. https://www.nyu.edu/classes/bello/FMT_files/9_MIDI_code.pdf. Accessed 14 May 2018