Fragments-Expert

User Manual for v1.2

Mehdi Teimouri and Zahra Seyedghorban

Information Theory and Coding Laboratory

University of Tehran

Fall 2020

Preface

File fragment classification is an important step in digital forensics. Most of the researches in this area have focused on content-based file fragment classification. Many research projects in content-based file fragment classification, extract some feature such as n-gram, Shannon entropy, byte frequency distribution, etc. Then, a decision machine model is trained using these features. There is a little amount of publicly available computer codes in this area. Fragments-Expert is a software package that tries to fill this void by providing the researchers with many feature extraction methods and machine learning algorithms in the field of file fragment classification. This document provides the user manual for Fragments-Expert v1.2. The software can be downloaded at https://github.com/mehditeimouri-UT/Fragments-Expert/releases/tag/1.2. To run Fragments-Expert you need 64-bit MATLAB R2015b or newer releases for Windows.

Contents

1 Introduction

Fragments-Expert is an open-source software package for feature extraction from file fragments and classification of file fragments into different file formats. Fragments-Expert is well equipped for fragment generation, feature extraction/selection, fragment classification, and data visualization.

In the following sections, first, we briefly review the background of file format classification. Then, we describe the overall structure of Fragments-Expert software.

1-1 File Fragments Classification

The classification of file fragments of various file formats is an essential task in various applications such as digital forensics, virus detection, and firewall protection. Many researchers have tried different feature sets and methods to classify file fragments.

The most common and fastest way to identify file type is to check the file extension. This method is very unreliable and can be easily spoofed. Moreover, most of the time, the examiner is dealing with a file fragment and not the whole file itself.

Another method for file type identification is through magic bytes. Magic bytes are predefined sequences that exist in the file header. Some earlier works used these magic bytes as a signature to detect a file type. However, as mentioned earlier, we may not have access to the complete file or the header. Also, some file formats don't have these fixed sequences in their header.

The practical method of file type identification is to examine the contents of fragments. Many works have presented different features and methods for this task. This content-based approach for file type classification has brought much attention to itself. The focus of Fragments-Expert is on this method.

1-2 Fragments-Expert Software

Though the presented methods for file type classification have some promising results and evaluations, the community lacks a suitable tool that can integrate all these feature sets and methods. Fragments-Expert is a software package that tries to fill this void. Fragments-Expert gives the researchers the ability to implement different examinations. Since the features and methods are integrated into one package, multiple tests can be done expeditiously.

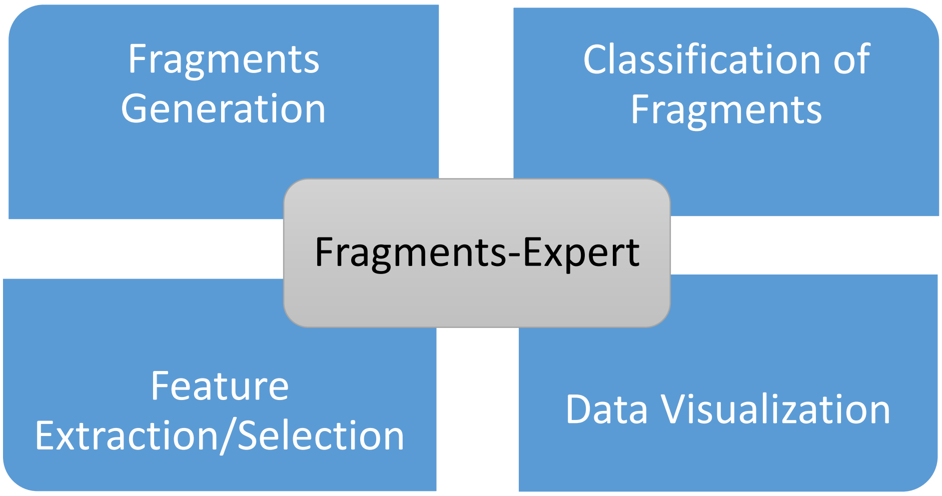


Figure 1-1: Four main modules of Fragments-Expert.

Figure 1-1 shows four main modules of Fragments-Expert: Fragments Generation, Feature Extraction/Selection, Classification of Fragments, and Data Visualization. In each module, multiple tools are available that enable the researchers to simply examine the effect of different feature types and classification methods.

- **Fragments Generation:** Using Fragments-Expert, you can generate fragments with specific parameters from raw multimedia files. This procedure is explained in section 2-1.
- **Feature Extraction/Selection:** Most common feature types are available in Fragments-Expert. In section 2-2, the list of different feature types and their descriptions are presented. Two common feature selection tools are also implemented in Fragments-Experts that are described in section 2-4. Various operations on datasets, such as randomization of Datasets, are available. These operations are explained in sections 2-5 to 2-8.
- **Classification of Fragments:** To classify file fragments into file formats, the user can employ various common machine learning methods. Training, test, and cross-validation are provided for each method. The processes of training, test, and cross-validation are explained in sections 3-1, 3-2, and 3-3, respectively.
- **Data Visualization:** This module offers some tools for visualizing data samples. Visualization tools are explained in section 4.

The remainder of this document is organized as follows. Section 2 deals with the dataset generation and manipulation. Generating fragments, feature extraction, feature selection, dataset permutation, merging datasets, merging class labels, and selecting sub-dataset are all described in this section. In Section 3, the processes of training, test, and cross-validation with available machine learning methods are explained. Data visualizing tools are covered in Section 4. Finally, in Section 5, loading previously generated datasets and results are explained.

2 Dataset Generation

Dataset generation and manipulation is the main capability of Fragments-Expert. You can generate a dataset of fragments from raw multimedia files in the form of generic binary data files. After that, these binary files can be used to generate a dataset of file fragment features. Many feature types, including all commonly used features, are available in the software. You can also generate a dataset for an already trained machine.

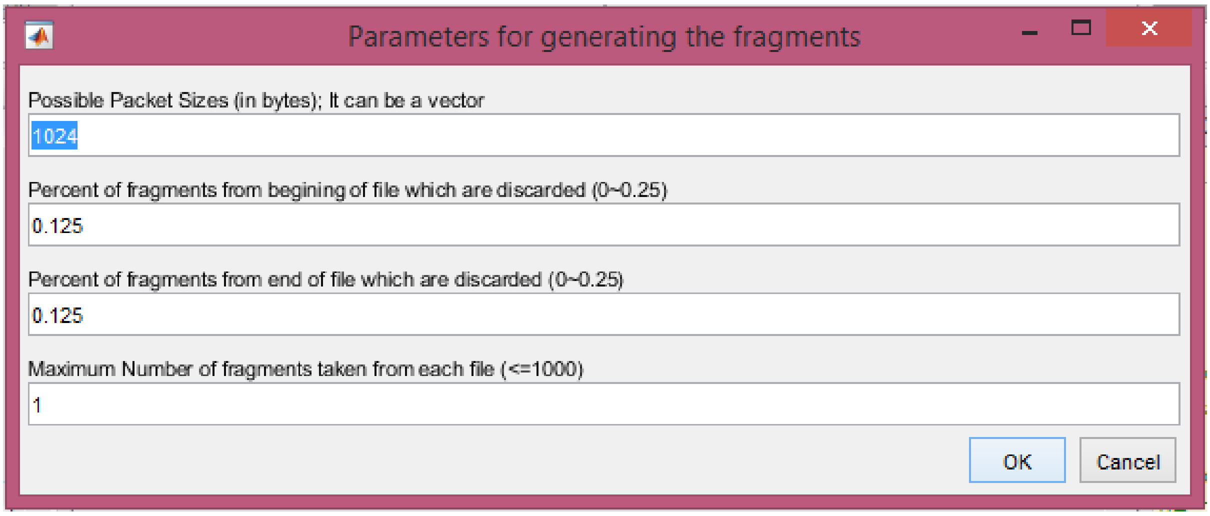
In the Fragments-Expert environment, you can apply some operations on a dataset including random permutation of samples in a dataset, merging two datasets, merging class labels within a dataset, and generating sub-dataset from a dataset.

Two methods for feature selection are also provided. The first method is an embedded method for feature selection that uses a decision tree to select features. The second method is a wrapper method that uses linear discriminant analysis (LDA) to obtain better features for classification using a forward feature selection strategy.

2-1 Generating Dataset of File Fragments

To extract fragments from raw multimedia files, click on the “Convert Raw Multimedia to Fragments Dataset” submenu in the “Dataset” menu. By clicking on this item, a new window, as shown in Figure 2-1, will open. In this stage, you should choose the parameters for generating the fragments as follows.

- The first input is an integer (or a vector of integers) representing the size(s) of generated fragments in bytes. If you specify a vector, the generated fragments are with lengths that are taken randomly and independently from this vector.
- The second input determines the percent of fragments from the beginning of the file that should be discarded
- The third input determines the percent of fragments from the end of the file that should be discarded
- The last input is the maximum number of fragments taken from each file.



Parameters for generating the fragments

Possible Packet Sizes (in bytes); It can be a vector

1024

Percent of fragments from begining of file which are discarded (0~0.25)

0.125

Percent of fragments from end of file which are discarded (0~0.25)

0.125

Maximum Number of fragments taken from each file (<=1000)

1

Figure 2-1: Parameters for generating the fragments.

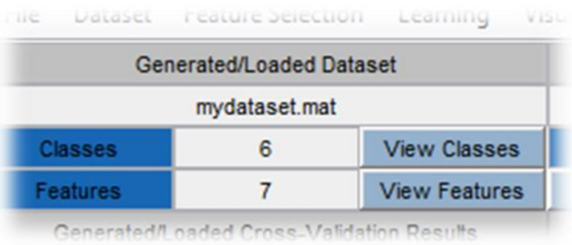
After filling the parameters in and hitting the OK button, you are prompted to select the main folder that contains the multimedia files. Note that this should be the directory that contains some folders, in which the raw files are stored. It is assumed that the file types in each subfolder are the same. After that, you must select the folder in which you want the fragments to be stored. Note that the second folder that you select should not be a subfolder of the first folder. In other words, you must choose a completely different directory or the action will be aborted. After that, the file fragments generation starts. Corresponding to each subfolder, a generic binary data file with .dat extension is created that contains the fragments taken from the files of that subfolder. The total number of fragments for each class (i.e. each subfolder) is shown in the main text window of the software.

2-2 Generating Dataset of Features

For generating a dataset of features from generic binary data files, click on the “Generate Dataset from Generic Binary Files of Fragments” submenu in the “Dataset” menu. In the next window, a list of feature types is shown. These features are listed below and explained in more detail in the following sections.

- Byte Frequency Distribution
- Rate of Change
- Longest Contiguous Streak of Repeating Bytes
- n–grams
- Byte Concentration Features: Low, ASCII, and High
- Basic Lower-Order Statistics: Mean, STD, Mode, Median, and MAD
- Higher-Order Statistics: Kurtosis and Skewness
- Bicoherence
- Window-Based Statistics
- Auto-Correlation
- Frequency Domain Statistics (Mean, STD, Skewness)
- Binary Ratio
- Entropy
- Video Patterns
- Audio Patterns
- Kolmogorov Complexity
- False Nearest Neighbors
- Lyapunov Exponents
- GIST Features
- Longest Common Subsequence
- Longest Common Substring
- Centroid Model

From the displayed list, select the features that you want to be extracted from your fragments. You can multi-select from the list using the Ctrl key. After that, hit the OK button. If your selected features require any parameters, another window will be opened to fill in the parameters. Then you must select the generic binary data files containing the fragments. Each file in .dat format represents a category (i.e. a class label). A progress bar is shown during reading the fragments. After reading the fragments, the software wants you to confirm variable names corresponding to class labels. Here you can choose any desired name for each category. After confirming the names, the feature extraction process starts. Once, this process is completed, you should save the generated dataset. The dataset will be loaded in the software environment under the “Generated/Loaded Dataset” section (see Figure 2-2); from there, you can view the classes and features. The number of data samples in each class is equal to the number of fragments in the corresponding .dat file.



Generated/Loaded Dataset

mydataset.mat

Classes 6

Features 7

Figure 2-2: An example of a generated dataset with six classes and seven features in the software environment.

2-2-1 Byte Frequency Distribution

This feature type gives you a vector with length 260. The first 256 values are the byte frequency distribution (BFD) values. These values show the normalized frequency of byte values 0 to 255. BFD_i is the normalized frequency of byte value $i = 0,1, \dots ,255$,

$$BFD_i = p_i \times 256, \tag{2-1}$$

where p_i is the estimated probability of each byte value that is calculated as

$$p_i = \frac{f_i}{L}, \tag{2-2}$$

in which f_i is the frequency of ith byte value and L is the length of the fragment.

The other four features computed along with BFD are as follows. The first three features are defined in [1].

- SdFreq: Standard deviation of byte frequencies.
- ModesFreq: The sum of the four highest byte frequencies
- CorNextFreq: Correlation of the frequencies of byte values j and j+1
- ChiSq: The chi-square goodness of the fit test is a measure of randomness. Here we are using this measurement to compare the observed distribution of byte values to a uniform random distribution. For each byte value, we calculate the sum of the squared differences between the observed frequency of the byte values in the fragment and the expected frequency of the byte values in a uniform random distribution divided by the expected frequencies as follows. [2]

$$T = \sum_{i=0}^{n=1} \frac{(f_i - \frac{L}{256})^2}{\frac{L}{256}}. \quad \textbf{(2-3)}$$

Then we use `chi2cdf(T,255,'upper')` command to compute $p - value$ for this test.

2-2-2 Byte Bigrams

Bigram counts of bytes consider pairwise counts of consecutive bytes in a fragment. To calculate bigrams, first, all possible pairs of byte values must be counted, so this feature type is a 65,536-dimensional vector containing normalized frequencies for byte pairs.

2-2-3 Rate of Change

Rate of change (RoC) is the vector of consist of the frequencies of the absolute values of the differences between consecutive byte values in a file fragment. If we denote a fragment of bytes with length L by $\mathbf{x} = [x_1, x_2, \ldots, x_L]$, the jth element of RoC vector, $j = 0, 1, \ldots, 255$, is calculated as follows.

$$R_j = \frac{\sum_{|x_i - x_{i+1}| = j} 1}{L - 1}. \quad \textbf{(2-4)}$$

Then each element of the RoC vector is normalized as follows.

$$\hat{R}_j = \begin{cases} \frac{256 \times 256}{2 \times (256 - j)} \times R_j & j \neq 0 \\ 256 \times R_j & j = 0 \end{cases}. \quad \textbf{(2-5)}$$

When the user includes this feature type in the feature extraction process, the mean of the rate of changes is also calculated as a feature. So, in this case, the feature vector is a vector with length 257.

2-2-4 Longest Contiguous Streak of Repeating Bytes

This feature is the normalized length of the longest contiguous streak of repeating bytes in the fragment. For example, for vector

$$[12\ \ 123\ \ 123\ \ 123\ \ 43\ \ 123\ \ 43\ \ 43\ \ 43\ \ 43\ \ 43\ \ 123\ \ 43\ \ 76\ \ 54\ \ 54\ \ 54\ \ 54],$$

the byte value 43 is repeated consecutively five times. So, in this case, the output feature is 5/18.

2-2-5 *n*-grams

In this case, the frequencies of the bit sequences with length n are calculated. This feature is defined for bitstreams. So, the fragment of bytes is first converted to its equivalent binary form. As an example, consider the binary vector [1 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0] with length 16. To calculate *2*-grams for this vector, first, all possible combinations of zeros and ones with length two must be counted. For example, the bit pattern 0 0 is repeated three times. So, the normalized 2-gram value for this pattern is equal to $\frac{3}{16-2+1} = 0.2$.

When you include this feature type in the feature extraction process, you must choose the n values in the next window. You can enter a vector of integer values for *n*. In this case, *n*-grams for all the values in this vector will be calculated. In the current version, *n* can be any integer value from 1 to 13.

2-2-6 Byte Concentration Features

From the BFD vector described in section 2-2-1, these three features are calculated [1]:

- Low: Sum of the BFD_i values for $0 \leq j < 32$.
- ASCII: Sum of the BFD_i values for $32 \leq j < 128$.
- High: Sum of the BFD_i values for $192 \leq j < 256$.

2-2-7 Basic Lower-Order Statistics

By selecting this feature type, seven different statistics are calculated as follows. First, three different mean values, i.e. arithmetic, geometric, and harmonic, are calculated as follows.

$$\mu_A = \frac{1}{L} \sum_{i=1}^{L} x_i, \tag{2-6}$$

$$\mu_G = \left(\prod_{i=1}^{L} x_i \right)^{\frac{1}{L}}, \tag{2-7}$$

and

$$\mu_H = \frac{L}{\sum_{i=1}^{L} \frac{1}{x_i}}. \tag{2-8}$$

Moreover, the standard deviation (STD) is calculated as follows.

$$\sigma = \sqrt{\frac{1}{L-1} \sum_{i=1}^{L} |x_i - \mu_A|^2}. \tag{2-9}$$

The mode value, which is the most frequently occurring value in **x**, is also calculated. The median value, which is the median value among all values of **x**, is also calculated. Finally, the MAD value, which is the mean absolute deviation of values, is calculated as $mean\left(abs\left(\mathbf{x} - mean(\mathbf{x})\right)\right)$.

2-2-8 Higher-Order Statistics

The unbiased estimation of kurtosis and skewness is employed in Fragments-Expert. Kurtosis estimation is calculated as follows

$$kurtosis = \frac{L-1}{(L-2)(L-3)}\left((L+1)K - 3(L-1)\right) + 3, \tag{2-10}$$

where K is

$$K = \frac{\frac{1}{L}\sum_{i=1}^{L}(x_i - \mu_A)^4}{(\frac{1}{L}\sum_{i=1}^{L}(x_i - \mu_A)^2)^2}. \tag{2-11}$$

Moreover, skewness estimation is calculated as follows

$$skewness = \frac{\sqrt{L(L-1)}}{L-2} S, \quad \textbf{(2-12)}$$

where S is

$$S = \frac{\frac{1}{L}\sum_{i=1}^{L}(x_i - \mu_A)^3}{(\sqrt{\frac{1}{L}\sum_{i=1}^{L}(x_i - \mu_A)^2})^3}. \quad \textbf{(2-13)}$$

2-2-9 Bicoherence

Bicoherence is a higher-order statistic that measures the non-linearity and non-Gaussianity in a given fragment of byte values. Here we computed the average bicoherence according to [3].

2-2-10 Window-Based Statistics

These features are defined according to [4]. For calculating these features you must at first choose a window size. The default value is $W = 256$.

Five statistics including delta moving average, delta^2 moving average, delta standard deviation, delta^2 standard deviation, and deviation from the standard deviation are obtained from the values in the file fragment using non-overlapping and consecutive windows.

The moving average is calculated by taking the average of the mean of byte values in all windows. If $\mu_j;\ j = 1,2,\dots,J = \left\lfloor\frac{L}{W}\right\rfloor$ denotes the average of byte values in window j, the delta moving average is calculated as follows

$$\Delta\mu = \frac{1}{J-1}\sum_{j=1}^{J-1}\left|\mu_{j+1} - \mu_j\right|. \quad \textbf{(2-14)}$$

Moreover, delta^2 moving average is calculated as follows

$$\Delta\Delta\mu = \frac{1}{J-2}\sum_{j=1}^{J-2}\left|\left|\mu_{j+2} - \mu_{j+1}\right| - \left|\mu_{j+1} - \mu_j\right|\right|. \quad \textbf{(2-15)}$$

Similarly, if $\sigma_j; j = 1,2,\dots,J$ denotes the standard deviation of byte values in window j, delta standard deviation, and delta^2 standard deviation are respectively calculated as follows

$$\Delta\sigma = \frac{1}{J-1}\sum_{j=1}^{J-1}\left|\sigma_{j+1} - \sigma_j\right|, \quad \textbf{(2-16)}$$

$$\Delta\Delta\sigma = \frac{1}{J-2}\sum_{j=1}^{J-2}\left|\left|\sigma_{j+2} - \sigma_{j+1}\right| - \left|\sigma_{j+1} - \sigma_j\right|\right|. \quad \textbf{(2-17)}$$

Finally, the deviation from the standard deviation is calculated as follows

$$d_\sigma = \frac{1}{J}\sum_{i=1}^{J}\left|\sigma_j - \sigma\right|. \quad \textbf{(2-18)}$$

2-2-11 Autocorrelation

This feature is the sample autocorrelation of the input data fragment up to a specific lag value ℓ. The sample autocorrelation measures the correlation between x_t and x_{t+k}, where $k = 1,\dots,\ell$. The sample autocorrelation for lag k is

$$r_k = \frac{c_k}{c_0}, \quad \textbf{(2-19)}$$

where

$$c_k = \frac{1}{L-1}\sum_{j=1}^{L-k}\left(x_j - \mu_A\right)\left(x_{j+k} - \mu_A\right). \tag{2-20}$$

Note that $c_0 = \sigma^2$ is the sample variance of the fragment. By choosing this feature, you have to specify the maximum lag value in the next window.

2-2-12 Frequency Domain Statistics

As represented in [5], the frequency spectrum of a vector (in or context, a fragment) is divided into equal sub-bands. Then the features of mean, variance, and skewness are extracted for each sub-band. For example, if the number of sub-bands is equal to four, 12 frequency-domain statistics are extracted. You can set the number of sub-bands from 1 to 8.

2-2-13 Binary Ratio

The definition of Binary Ratio (BRO) is obtained from [6] and is calculated over the bitstream (i.e. the equivalent binary form of the fragment). Each byte value is converted to an 8-bit representation. If $\mathbf{b} = [b_1, b_2, \dots, b_{8\times L}]$ denotes the equivalent binary form of the fragment, the Binary Ratio is defined as follows.

$$\text{BRO} = \frac{\sum_{j=1}^{L}(1-b_i)}{\sum_{j=1}^{L} b_i}. \tag{2-21}$$

2-2-14 Entropy

In this case, two statistics are obtained for the fragment: entropy and the difference between L-truncated entropy of uniform distribution [7] and the value of obtained entropy.

Entropy in information theory is a measure of the uncertainty of a sequence. In our case, in which we have a fragment with length L, the Shannon entropy is defined as

$$H = -\sum_{i=0}^{255} p_i \log_2 p_i \tag{2-22}$$

The L-truncated entropy of a uniform distribution can be approximated as [7]

$$H_N(U) \cong \log m + \log c - e^{-c}\sum_{j=1}^{+\infty}\frac{e^{j-1}}{(j-1)!}\log j, \tag{2-23}$$

where m is the number of possible byte values (which, in this case, is 256) and $c = \frac{L}{m}$.

2-2-15 Video Patterns

Some video formats have repeating patterns in their payload. The occurrence rate of these patterns can be used to determine the format of a file fragment. Here, we count the occurrences of these patterns and to make the resulting number independent of the length of the fragment and also the length of the pattern, we normalize it as follows.

$$\frac{frq}{L - l_p + 1} \times 2^{8\times l_p}, \tag{2-24}$$

where frq is the number of occurrences of the pattern in the fragment and l_p is the length of the pattern in bytes. Here, 17 patterns are considered. These patterns correspond to five different video formats. So, in this case, the feature vector contains 17 normalized pattern frequencies. In Table 2-1, employed video patterns are presented. The patterns for MKV, AVI, RMVB, and MP4, are proposed in [8].

Table 2-1: Different byte patterns for video formats presented in hexadecimal format.

Format	Pattern(s)
MKV	0xA0 0xA3
AVI	0x30306463 0x30317762
RMVB	0x0000 0x0001
OGV	0x4F676753
MP4	0x419A 0x019E 0x019F 0x419B 0x6742 0x419E 0x419F 0x6588 0x68CE 0x6588

2-2-16 Audio Patterns

This feature type calculates the normalized frequencies of specific audio bit patterns (i.e. sync words) in a file fragment. First, each byte value is converted to an 8-bit representation. Then, similar to the calculation of video patterns, normalization is applied to make the count of the occurrences independent of the length of the fragment and also the length of the pattern. In Table 2-2, the employed audio patterns are presented.

Table 2-2: Different bit patterns for audio formats [9].

Format	Pattern
MP3	1111 1111 1111
FLAC	1111 1111 1111 10

2-2-17 Kolmog*o*rov Complexity

The Kolmog***o***rov complexity of a fragment is a measure of the computational resources needed to specify the fragment. But, no general algorithm can determine this complexity. We employ the method of [10] for estimating this complexity. For the sake of run-time speed, this function is written in C-MEX format.

2-2-18 False Nearest Neighbors and Lyapunov Exponents

Two chaotic features false neighbors fraction (FNF) and Lyapunov exponent (LE) are implemented in Fragments-Expert. Hicsonmez et al employed chaotic features for the identification of audio codecs [5]. The concept of the chaotic features is based on the neighborhood of the signal vectors. Let’s define the signal vector $\mathbf{s_i} = [x_i, x_{i+1}, \dots, x_{i+(D-1)}]$, where D is the embedding dimension of the phase space. The distance between two nearest neighbors is defined as

$$d_D(\mathbf{s}_i, \mathbf{s}_j) = \sqrt{\sum_{k=0}^{D-1} (x_{i+k} - x_{j+k})^2}, \tag{2-25}$$

where $\mathbf{s}_j$ is the nearest neighbor of $\mathbf{s}_i$ on a nearby trajectory. If $d_D(\mathbf{s}_i, \mathbf{s}_j)$ is significantly different from $d_{D+1}(\mathbf{s}_i, \mathbf{s}_j)$, then $\mathbf{s}_i$ and $\mathbf{s}_j$ are considered to be a pair of false neighbors. After labeling all neighbors as true or false, the FNF is defined as the ratio of false neighbors to all neighbors [5]. So, we can define the feature vector F_D with three components: the fraction of false neighbors, the average size of the neighborhood, and the root mean squared (RMS) size of the neighborhood.

$$F_D = \{FNF, mean\left(d_D(\mathbf{s}_i, \mathbf{s}_j)\right), RMSE\left(d_D(\mathbf{s}_i, \mathbf{s}_j)\right)\} \tag{2-26}$$

The Lyapunov exponent is a chaotic feature that quantifies the predictability of a signal. A system with a greater magnitude of LE is said to be more unpredictable [5]. The LE is calculated for each embedding dimension D as

$$\lambda_D = \lim_{K \to +\infty} \frac{1}{K} \sum_{i=1}^{K} \log \frac{d_D(\mathbf{s}_{i+1}, \mathbf{s}_{j+1})}{d_D(\mathbf{s}_i, \mathbf{s}_j)}. \tag{2-27}$$

When you choose this feature type, you have to provide three parameters: the ratio factor, which is used to determine the false neighbors, and the minimum and maximum values for embedding dimensions. For the sake of run-time speed, this function is written in C-MEX format.

2-2-19 GIST Features

Ming Xu et al expressed that a vector of byte values (i.e. a fragment) can be reshaped into a matrix and regarded as a grayscale image [11]. So, they used GIST Descriptor that performs very good in scene and object classification [12]. To transform a fragment to a grayscale image and extracting GIST features, some parameters have to be set: Image row size, the number of non-overlapping windows in each dimension, which we denote by M, and the number of orientations for each scale. The last parameter is a vector of integer values which we denote by $[O_1, O_2, \dots]$. The result of this type of feature extraction is a feature vector with length $M^2 \sum_i O_i$

2-2-20 Longest Common Substrings and Longest Common Subsequences

We define the longest common substring of two fragments to be the longest byte pattern existing in both fragments. A subsequence of a byte stream is any sequence of bytes obtained by deletions from the original. Accordingly, we can define the longest common subsequence of two fragments.

To define the longest common substring and the longest common subsequence features, we need certain fragments, which are called representatives. For each fragment in the dataset, the longest common substring and the longest common subsequence between that fragment and each representative are calculated. When using these features you have to select which class you want the representatives to be obtained from and where in the dataset the representatives should be taken (i.e. from the beginning of the dataset, end of the dataset, or random positions in the dataset). Also, you must input the number of representative samples for the selected class. Corresponding to each representative class, two features are computed for a fragment: the average length of the longest common substrings and the average length of the longest common subsequences. For the sake of run-time speed, this function is written in C-MEX format.

2-2-21 Centroid Models

Using a class of representatives, you can obtain a measure of similarity between the byte frequency distribution of that class instances and the investigated fragment. The value of similarity can then be used as a feature for classification.

In Fragments-Expert, we have implemented two similarity measures: cosine similarity and Mahalanobis distance. When you select this feature type, you are asked to select the classes of representatives for centroid models among all class labels. Then you must enter the number of representatives for each class and where in the dataset the representatives should be taken.

After the parameters are set, BFD of the representative instances is calculated. Then for each byte value of the BFD, the mean and the standard deviation values are calculated. This mean and standard deviation values form a centroid model. By denoting the mean and the standard deviation BFD values of the representative instances by μ_i^c and σ_i^c $(i = 0,1,\dots,255)$, we calculate the similarity features for each investigated fragment using

$$\text{CosineSimilarity} = \frac{\sum_i (BFD_i \times \mu_i^c)}{\sqrt{\sum_i BFD_i{}^2} \times \sqrt{\sum_i \mu_i^{c\,2}}} \tag{2-28}$$

and

$$\text{MahalanobisDistance} = \sqrt{\sum\nolimits_i \frac{(BFD_i - \mu_i^c)^2}{(0.01 + \sigma_i^c)}}\,, \tag{2-29}$$

where BFD_i is the normalized frequency of byte values $i = 0,1,\dots,255$ in the examined fragment.

2-3 Generate Dataset for Decision Machine

You may have a previously trained decision machine and now you want to generate a new dataset that its feature set is matched to this already trained decision machine. To do that, first, the previously trained machine must be loaded in the software (see Section 5-2). Then click on the “Generate Dataset (for Decision Machine) from Generic Binary Files of Fragments” submenu in the “Dataset” menu.

The next steps are quite similar to the steps explained in section 2-2; except now you don’t need to select any feature type. The dataset is generated automatically and then, after saving on the disk, it is loaded in the software environment.

2-4 Feature Selection

When you create a dataset of features, you may have many features that most of them might be useless in the classification process. To obtain more relevant features and achieve less computational cost in training decision machines, we need to employ some feature selection methods. Sometimes as an expert in the field, you can determine the features that contribute most to your classification scenario. However, manually selecting the features could be very hard or time-consuming. In Fragments-Expert, we have implemented four useful feature extraction methods.

2-4-1 Embedded Method

Embedded methods refer to techniques that select the features during the training phase. The learning algorithm itself selects the features as a step of the learning. The most typical embedded method is the decision tree.

In each training step, a decision tree selects the best feature based on a goodness measure. So, as we go upper in a trained tree, the decisions are made based on more relevant features. We can take advantage of this property and turn the decision tree into a feature selector.

To use this method, first, the dataset must be loaded in the software (see Section 5-1). Then click on the “Embedded: Decision Tree” submenu in the “Feature Selection” menu. Since you are about to train a decision tree, you must input the parameters similar to the parameters described in Section 3-1-1. After the parameters are set and the training phase is completed, you will see a list of selected features sorted based on the relative node sizes. You can select among these features. Finally, the new feature set is created and loaded in the software after saving it on the disk.

2-4-2 Filter Mehtod

Filter methods measure the degree to which a feature is relevant to class labels. In Fragments-Expert, we use Pearson's Linear Correlation Coefficient to find the correlation between the features and the class labels.

To use this method, first, the dataset must be loaded in the software (see Section 5-1). Then click on the “Filter: Pearson Correlation Coefficient” submenu in the “Feature Selection” menu. Then a list of features is shown, sorted by their correlation. You can select among these features to build and then save the new dataset.

2-4-3 Wrapper Method

Wrapper methods search through different possible subsets of the feature set. These methods evaluate each subset by the accuracy of the learning algorithm. Note that for large problems, this method is very time-consuming.

In Fragments-Expert we have employed linear discriminant analysis as the core learning algorithm for this type of feature selection method. To apply this method, first, load the dataset in the software environment. Then, click on the “Wrapper: Sequential Forward Selection with LDA” submenu in the “Feature Selection” menu. LDA is used with cross-validation in a forward selection scenario to select the features. To get an understanding of LDA and cross-validation refer to Sections 3-1-6 and 3-3. During the feature selection process, the included features and corresponding accuracies are listed in the software.

2-4-4 Feature Transformation

In general, feature transformation is the process of mapping the feature values into a new space. In machine learning, feature transformation techniques are often used to reduce the dimensionality of datasets. Principal Component Analysis (PCA) is a frequently used feature selection method in this category. It computes the principal components of data (i.e. the eigenvectors of the data's covariance matrix) and uses a subset of them to perform a change of basis on the data. PCA projects the data points onto a lower-dimensional space.

To use this method, first, the dataset must be loaded in the software (see Section 5-1). Then click on the “Feature Transformation: Principal Component Analysis (PCA)” submenu in the “Feature Selection” menu. After the feature selection process is completed a list of features is shown that are sorted by their eigenvalues. You can select among these features to build and then save the new dataset.

2-5 Random Permutation of a Dataset

This process randomly permutes the samples in a dataset. Each sample contains the features that describe a specific fragment. Note that the permutation is performed in a way that the fragments of a single file stay together.

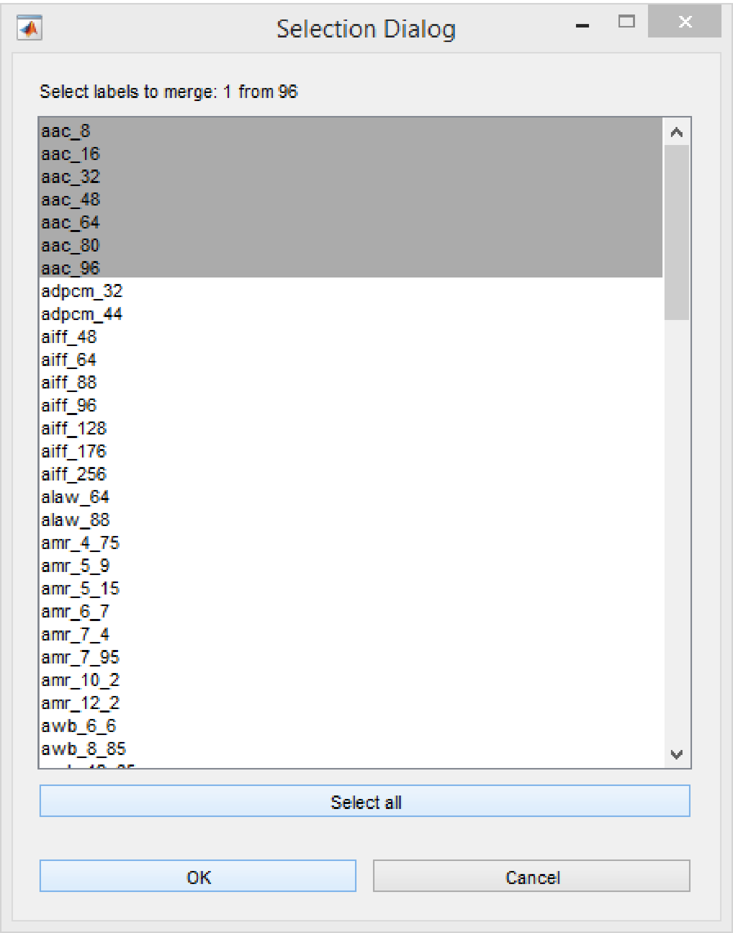
To permute the samples in a dataset, first, the dataset must be loaded in the software. Then, you should click on the “Random Permutation of Dataset” submenu in the “Dataset” menu. Then you must save the permuted dataset.

2-6 Merge Two Datasets

This part of Fragments-Expert helps you to add features to an already created dataset. In this case, the dataset with the new feature set of the same fragments must be loaded in the software. To add the features of this loaded dataset to an already saved dataset, click on the “Expand Dataset” submenu in the “Dataset” menu. Then you must select the old dataset. Note that both datasets must have matching dataset sizes, output labels, and file identifiers.

2-7 Merge Labels in a Dataset

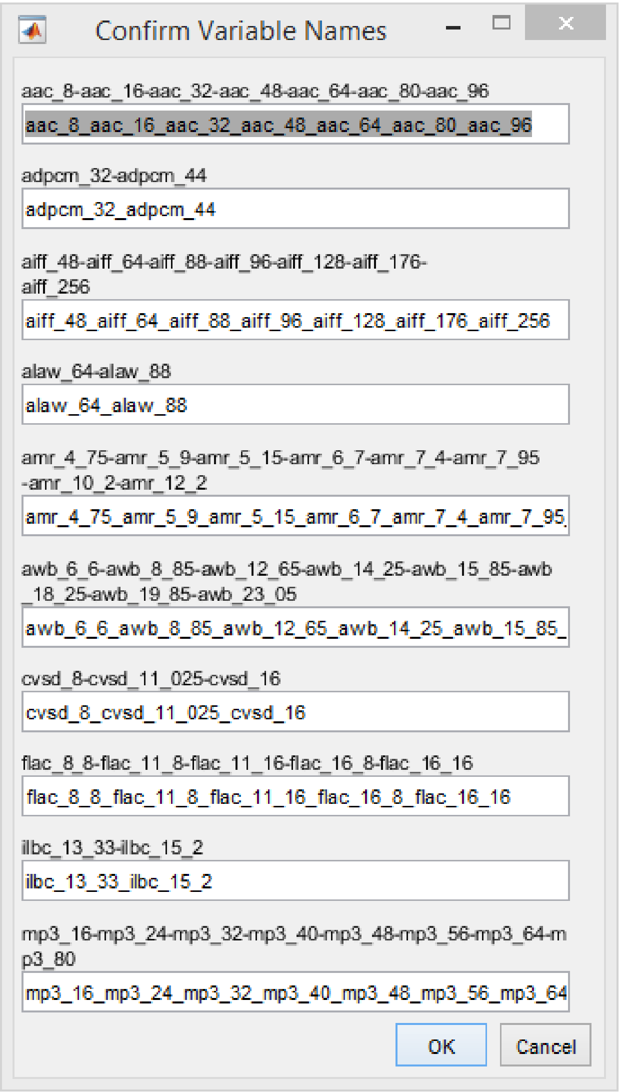
Sometimes you may want to merge labels in a dataset to form more general class labels. To do so, the dataset must be loaded first. Then click on the “Merge Labels in Dataset” submenu in the “Dataset” menu. A window will open. There you can multi-select the classes you want to merge their labels. If this process needs to be done for other classes click on the OK button, if not click on the cancel button. In Figure 2-3, an example is shown. In this example, we are merging the samples of AAC codec with different bitrates into one general label AAC. After selecting the merged labels, you need to confirm the labels for the merged classes. In Figure 2-4, an example is shown.



Select labels to merge: 1 from 96

aac_8
aac_16
aac_32
aac_48
aac_64
aac_80
aac_96
adpcm_32
adpcm_44
aiff_48
aiff_64
aiff_88
aiff_96
aiff_128
aiff_176
aiff_256
alaw_64
alaw_88
amr_4_75
amr_5_9
amr_5_15
amr_6_7
amr_7_4
amr_7_95
amr_10_2
amr_12_2
awb_6_6
awb_8_85

Figure 2-3: An example of selecting seven different labels to merge them into one class label.



aac_8-aac_16-aac_32-aac_48-aac_64-aac_80-aac_96
aac_8_aac_16_aac_32_aac_48_aac_64_aac_80_aac_96

adpcm_32-adpcm_44
adpcm_32_adpcm_44

aiff_48-aiff_64-aiff_88-aiff_96-aiff_128-aiff_176-
aiff_256
aiff_48_aiff_64_aiff_88_aiff_96_aiff_128_aiff_176_aiff_256

alaw_64-alaw_88
alaw_64_alaw_88

amr_4_75-amr_5_9-amr_5_15-amr_6_7-amr_7_4-amr_7_95
-amr_10_2-amr_12_2
amr_4_75_amr_5_9_amr_5_15_amr_6_7_amr_7_4_amr_7_95

awb_6_6-awb_8_85-awb_12_65-awb_14_25-awb_15_85-awb
_18_25-awb_19_85-awb_23_05
awb_6_6_awb_8_85_awb_12_65_awb_14_25_awb_15_85_

cvsd_8-cvsd_11_025-cvsd_16
cvsd_8_cvsd_11_025_cvsd_16

flac_8_8-flac_11_8-flac_11_16-flac_16_8-flac_16_16
flac_8_8_flac_11_8_flac_11_16_flac_16_8_flac_16_16

ilbc_13_33-ilbc_15_2
ilbc_13_33_ilbc_15_2

mp3_16-mp3_24-mp3_32-mp3_40-mp3_48-mp3_56-mp3_64-m
p3_80
mp3_16_mp3_24_mp3_32_mp3_40_mp3_48_mp3_56_mp3_64

Figure 2-4: An example of confirming labels for merged class labels.

2-8 Generating Sub-Datasets from a Dataset

This feature of Fragments-Expert helps you to select a subset of dataset classes and/or features. When the original dataset is loaded, click on the “Select Sub-Dataset” submenu in the “Dataset” menu. In the next window, you can multi-select the classes you want to keep. Then you can select among features. For both classes and features, if you want to keep all of them, select all.

3 Using Machine Learning for File Fragments Classification

Using Fragments-Expert, you can apply several machine learning algorithms and methods on your dataset. You can train, test, and cross-validate a model. The currently available machines are decision tree, support vector machine (SVM), random forest, ensemble *k*-nearest neighbors (*k*-NN), linear discriminant analysis, and neural network. In the following sections, more details are presented.

3-1 Train a Decision Machine

To train a decision model, first, your dataset must be loaded in the software environment. Then you should click on the “Train Decision Machine” submenu in the “Learning” menu. In the next window, you should select a decision model among decision machines. After selecting the decision model, you must provide some parameters. For each model, some default parameters are set; however, you may want to modify them to what works best for your experiment.

Three general parameters are required for all decision machines:

- The weighting method: This parameter determines whether the frequency of instances of each class label should be considered or not. If the weighting method “balanced” is chosen, the sample weights are set in a way that implies similar importance of all classes in the learning process. On the other hand, if the weighting method “uniform” is chosen, the classes with a larger number of samples are considered as more important classes.
- Start and end of the train/validation in the dataset: This parameter is a vector of length two with elements in the range of [0 1] that determines the relative position of the start and the end of the train/validation in the dataset. Note that 0 corresponds to the first sample in the dataset and 1 corresponds to the last sample in the dataset.
- Train and validation percentages taken from the dataset: This parameter is a vector of length two with a sum of elements equal to 100. The first element determines the percent of training data in the train/validation set. According to the software settings, at least 70% of the train/validation set should be dedicated to the training phase.

For most of the decision models, the method for feature scaling is also prompted. The scaling method can be either standardization (also called z-score) or min-max normalization. Assume that there we have S samples in the training phase, where each of them is defined by F features. For example, assume that $f_{i,j}; j = 1,2, \dots, S$ are the values of the features $f_i; i = 1,2, \dots, F$ over all samples. If we denote the scaled feature values by $\hat{f}_{i,j}$, for z-score scaling, we have

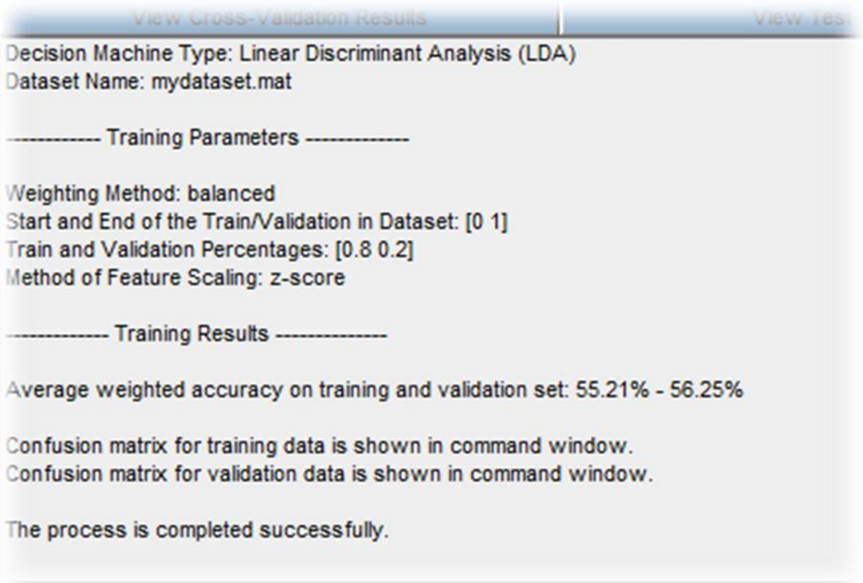
$$\hat{f}_{i,j} = \frac{f_{i,j} - \mu_{f,i}}{\sigma_{f,i}}, \tag{3-1}$$

where $\mu_{f,i}$ and $\sigma_{f,i}$ are respectively the mean and standard deviation for values $f_{i,j}; j = 1,2, \dots, S$. Furthermore, for min-max normalization we have.

$$\hat{f}_{i,j} = \frac{f_{i,j} - a_{f,i}}{b_{f,i} - a_{f,i}}, \tag{3-2}$$

where $a_{f,i}$ and $b_{f,i}$ are respectively the minimum and maximum values among $f_{i,j}; j = 1,2,\dots,S$.

After setting all parameters for training and pressing the OK button, a progress bar indicates the progression of the training process and the remaining time. After the training phase completion, the trained model is loaded in the software environment. Also, the training parameters and results are shown in the main text window of the software. The confusion matrices for training and validation are also shown in the command window. Figure 3-1 displays the results of an example with the LDA decision model.



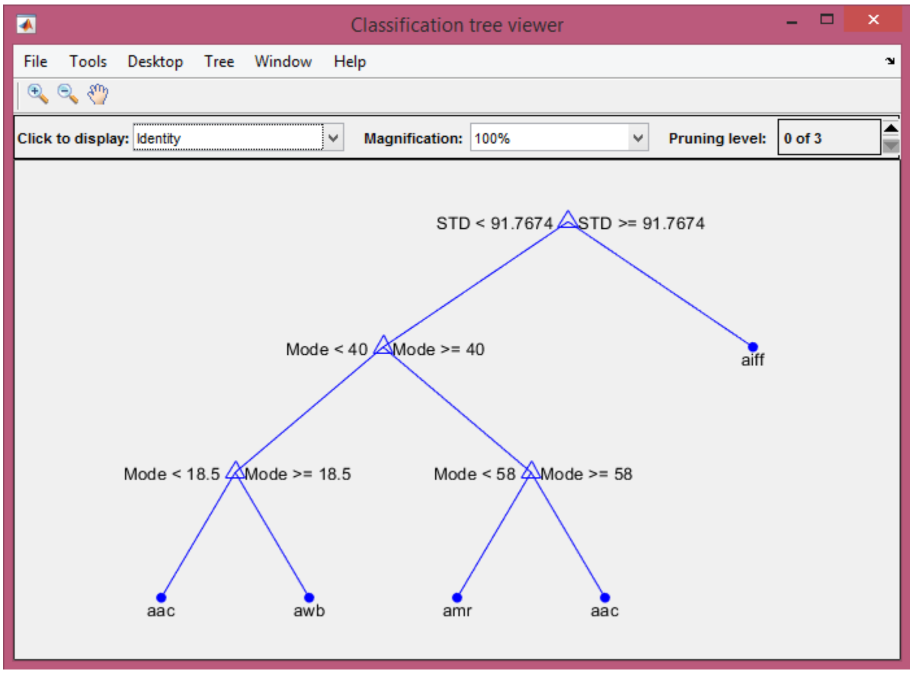
```
Decision Machine Type: Linear Discriminant Analysis (LDA)
Dataset Name: mydataset.mat

----------- Training Parameters -------------

Weighting Method: balanced
Start and End of the Train/Validation in Dataset: [0 1]
Train and Validation Percentages: [0.8 0.2]
Method of Feature Scaling: z-score

------------ Training Results --------------

Average weighted accuracy on training and validation set: 55.21% - 56.25%

Confusion matrix for training data is shown in command window.
Confusion matrix for validation data is shown in command window.

The process is completed successfully.
```

Figure 3-1: An example displaying training parameters and results in the main text window of the software.

3-1-1 Train a Decision Tree

To train a decision tree, besides the general parameters, you need to input the minimum relative number of leaf node observations to total samples. This relative number is then multiplied by the size of the training set. The result is the minimum number of observations per tree leaf. Note that the validation percent for the decision tree should be at least 15%.

When the training is completed, a figure window will open to display the trained tree. Also, you are prompted to save the trained model. In Figure 3-2 an example is shown.



Click to display: Identity

Magnification: 100%

Pruning level: 0 of 3

Figure 3-2: A very simple trained decision tree.

3-1-2 Train a Multi-Class Support Vector Machine Classifier

In Fragments-Expert, a multi-class SVM classifier is trained by a one-versus-all strategy, in which one binary classifier is trained per class. To train a multi-class support vector machine, besides the general parameters, you need to provide some other parameters. These parameters are as follows.

- Value for box constraint in SVM: This number should be a positive real number.
- Kernel function for SVM: You can choose rbf[1], linear, or polynomial.
- Polynomial order for polynomial kernel function: This number, which is denoted by q, should be an integer from 1 to 7.
- Value for scaling kernel of SVM: This number, which is denoted by c**,** should be a positive real number.

SVM algorithm assigns a box constraint to each observation in the training data. A box constraint is a parameter that controls the maximum penalty imposed on margin-violating observations. A kernel function is used to compute the elements of the Gram matrix. Each element of the Gram matrix is an inner product of the transformed predictors (i.e. features) using the kernel function. Suppose $G(\hat{\mathbf{f}}_j, \hat{\mathbf{f}}_k)$ is element (j, k) of the Gram matrix, where $\hat{\mathbf{f}}_j = [\hat{f}_{1,j}, \hat{f}_{2,j}, \dots, \hat{f}_{F,j}]$ and $\hat{\mathbf{f}}_k = [\hat{f}_{1,k}, \hat{f}_{2,k}, \dots, \hat{f}_{F,k}]$ are F-dimensional normalized feature vectors representing samples j and k in the training set. In Table 3-1, a brief description of kernel functions is given.

[1] Radial Basis Function

Table 3-1: Different SVM kernels.

Kernel Function	Description	Formula
rbf	Radial Basis Function	$G(f_j, f_k) = \mathrm{e}^{-\lVert \hat{\mathbf{f}}_j - \hat{\mathbf{f}}_k \rVert^2 / \mathrm{c}^2}$
linear	Linear	$G(f_j, f_k) = \hat{\mathbf{f}}_j \hat{\mathbf{f}}_k{}^T / \mathrm{c}^2$
polynomial	A polynomial with order q	$G(f_j, f_k) = (1 + \hat{\mathbf{f}}_j \hat{\mathbf{f}}_k{}^T / \mathrm{c}^2)^q$

3-1-3 Train a Random Forest

To train a Random Forest, besides the general parameters, you need to choose the number of trees in the random forest and the minimum relative number of leaf node observations to total samples in each tree. This relative number is then multiplied by the size of the training set. The result is the minimum number of observations per tree leaf.

3-1-4 Train an Ensemble of *k*-Nearest Neighbors Classifiers

To train an ensemble k-nearest neighbors classifiers, besides the general parameters, you need to provide some other parameters. These parameters are as follows:

- the number of randomly selected features for each k-NN learner,
- the number of k-NN learners in the ensemble,
- and the number of nearest neighbors for classifying each sample.

3-1-5 Train a Naïve Bayes Classifier

For this classifier, no additional input parameters are asked by the software. The naïve Bayes model is trained with the following default parameters.

- Data distribution: Kernel smoothing density estimate is used to model the data.
- Kernel smoother type: Gaussian is set as the kernel smoother type.

3-1-6 Train a Linear Discriminant Analysis Classifier

You can use LDA classifier to classify fragments based on their feature distribution. The model assumes data has a Gaussian mixture distribution. Since the discriminator in the current version of Fragments-Expert is pseudo-linear, the model assumes the same covariance matrix for each class, and only the means vary.

3-1-7 Train a Neural Network

You can train a neural network to classify training data. In the current version of Fragments-Expert, two-layer pattern recognition neural network model is considered. So, besides the general parameters, you need to input the dimension of the hidden layer.

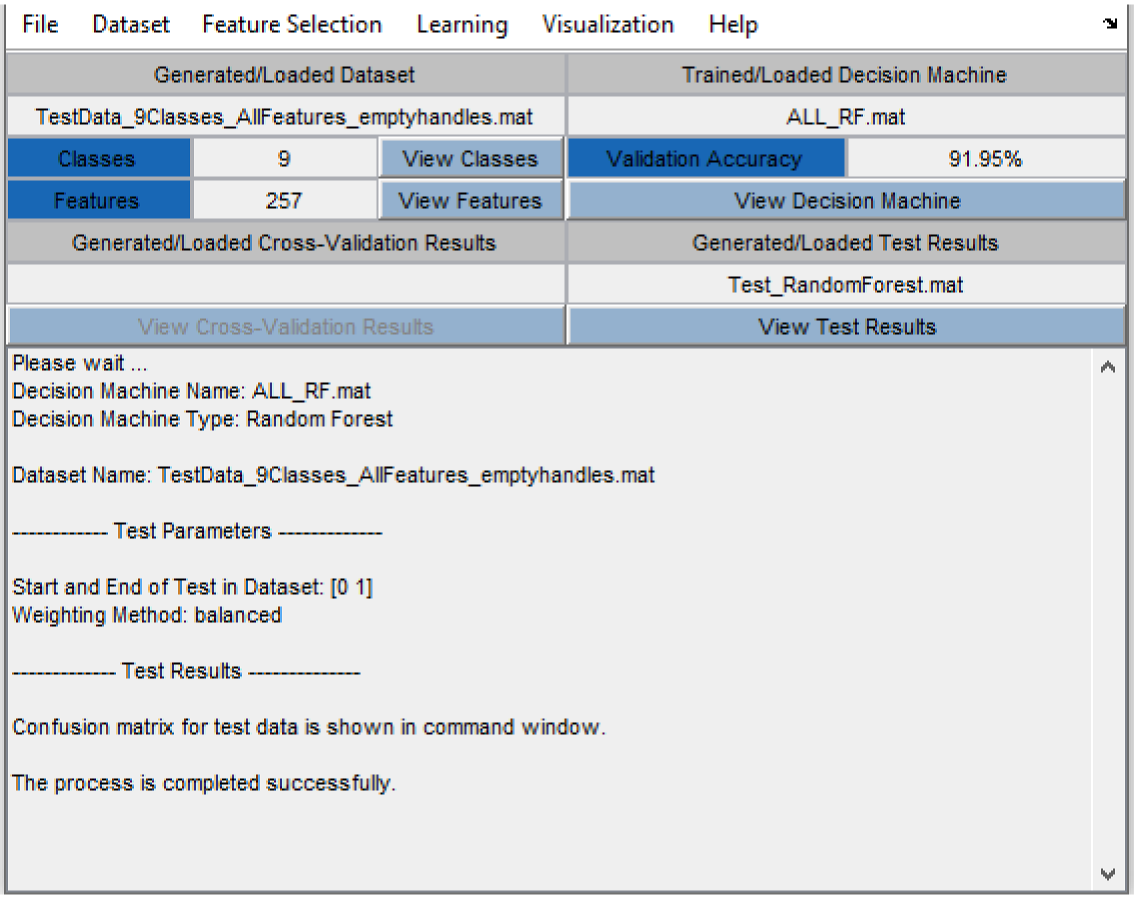
The training function updates weight and bias values according to the scaled conjugate gradient method. Moreover, cross-entropy is used as a measure for network performance.

3-2 Test a Trained Decision Machine

To test any trained model, the already trained machine and a compatible dataset must be loaded in the software environment. Note that the dataset must be compatible; i.e. the feature set of the dataset must be the same as the feature set used for training the model. You must provide the start and the end of the test samples in the dataset and also the weighting method.

After the test procedure is completed, the test result is saved. The test parameters and results are also shown in the main text window of the software.

Figure 3-3 is shown as an example. In this example, we have trained a random forest for an audio codec dataset. Now we want to test the performance of this trained random forest for classifying another nine file formats. We generate a dataset of features for nine file formats using the procedure explained in section 2-3.



Generated/Loaded Dataset: TestData_9Classes_AllFeatures_emptyhandles.mat

Trained/Loaded Decision Machine: ALL_RF.mat

Classes: 9

Features: 257

Validation Accuracy: 91.95%

Generated/Loaded Cross-Validation Results:

Generated/Loaded Test Results: Test_RandomForest.mat

```
Please wait ...
Decision Machine Name: ALL_RF.mat
Decision Machine Type: Random Forest

Dataset Name: TestData_9Classes_AllFeatures_emptyhandles.mat

------------ Test Parameters -------------

Start and End of Test in Dataset: [0 1]
Weighting Method: balanced

------------- Test Results --------------

Confusion matrix for test data is shown in command window.

The process is completed successfully.
```

Figure 3-3: The software environment after the test procedure.

3-3 Cross-Validation for a Decision Model

To obtain the optimal parameters for a decision model or to assess the average performance of a learning method, you can use cross-validation. To use cross-validation, first, your dataset must be loaded in the software environment. Then, you should click on the “Cross-Validation of Decision Machine” submenu in the “Learning” menu. In the next window, you can select among decision machines. Besides the parameters for each machine, you must choose the K value for K-fold cross-validation. When the procedure is completed, the cross-validation result is shown in the main text window of the software. The cross-validation confusion matrices are also shown in the command window of MATLAB.

4 Data Visualization

Visualization tools are available to give a better understanding of the distribution of feature values among data samples. In the current version of Fragments-Expert, you can use four powerful visualization tools.

4-1 t-Distributed Stochastic Neighbor Embedding (t-SNE)

You can display samples of a high-dimensional dataset in a low-dimensional space using t-Distributed Stochastic Neighbor Embedding (t-SNE). t-SNE is a nonlinear dimensionality reduction to embed data in a 2- or 3-dimensional space in a way that preserves neighbor identities. By centering a Gaussian on each data sample in the original space, a probability distribution is obtained in which similar samples are more likely to appear at a close distance. Using this distribution, the effective number of local neighbors (perplexity) must be chosen by hand [13].

To use this tool, click on the "t-Distributed Stochastic Neighbor Embedding (t-SNE)" submenu in the "Visualization" menu. In the next windows, you can select the class label that you want to visualize. To choose the next class, click on "OK". In each window, you can select more than one class to merge their samples. After confirming class labels, you must set/confirm some parameters: Final Reduced Dimensionality, Initial Reduced Dimensionality, The perplexity of the Gaussian kernel, and Maximum number of iterations. Figure 4-1 shows a 2-dimensional representation using t-SNE of three image file format samples.

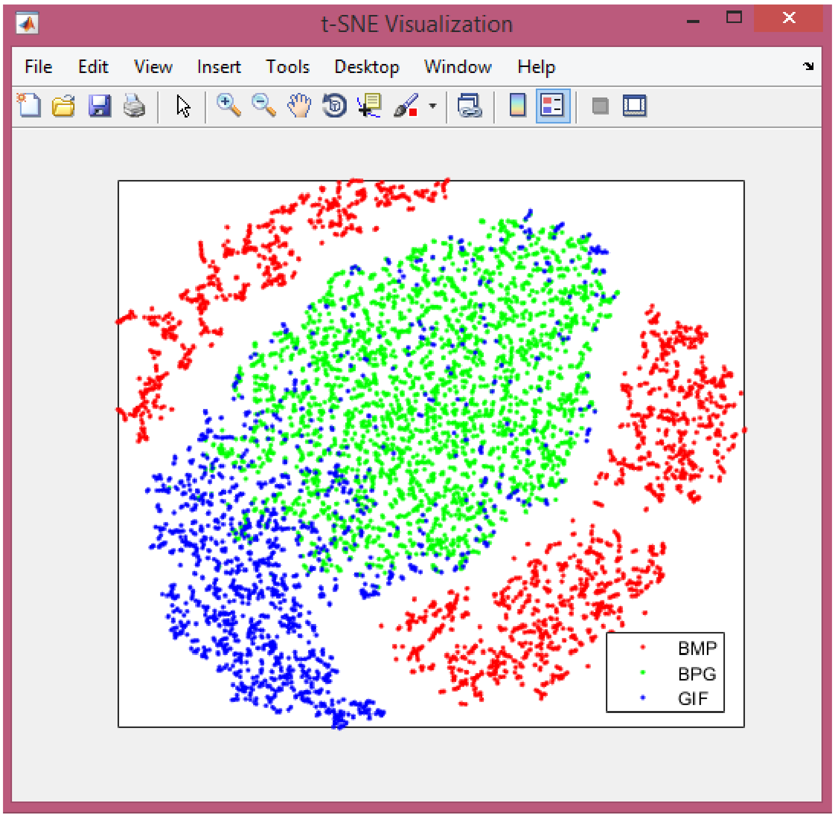


Figure 4-1: An example of a t-SNE representation.

4-2 Box Plot for Features

Box plot is a standard chart for depicting numerical data regarding its quartiles. Box plots show the five-number summary of a set of data, making it very convenient to understand its distribution. The five-number summary includes:

- Minimum score: The lowest data point, excluding outliers.
- Lower quartile: The median of the lower half of the data. 25% of the data falls below this value.
- Median: The mid-point of the data.
- Upper quartile: The median of the upper half of the data. 75% of the data falls below this value.
- Maximum score: The highest data point, excluding outliers.

To use this tool for visualization, click on the “Box Plot of Features” submenu in the “Visualization”' menu. In the next windows, you can select the feature(s) that you want to visualize. After selecting the features, choose the desired class labels. In each step, you can select more than one class label to merge their samples. After confirming the labels, you must choose the subplot organization. In Figure 4-2, box plots of two features (mode and median) for two image file formats are shown.

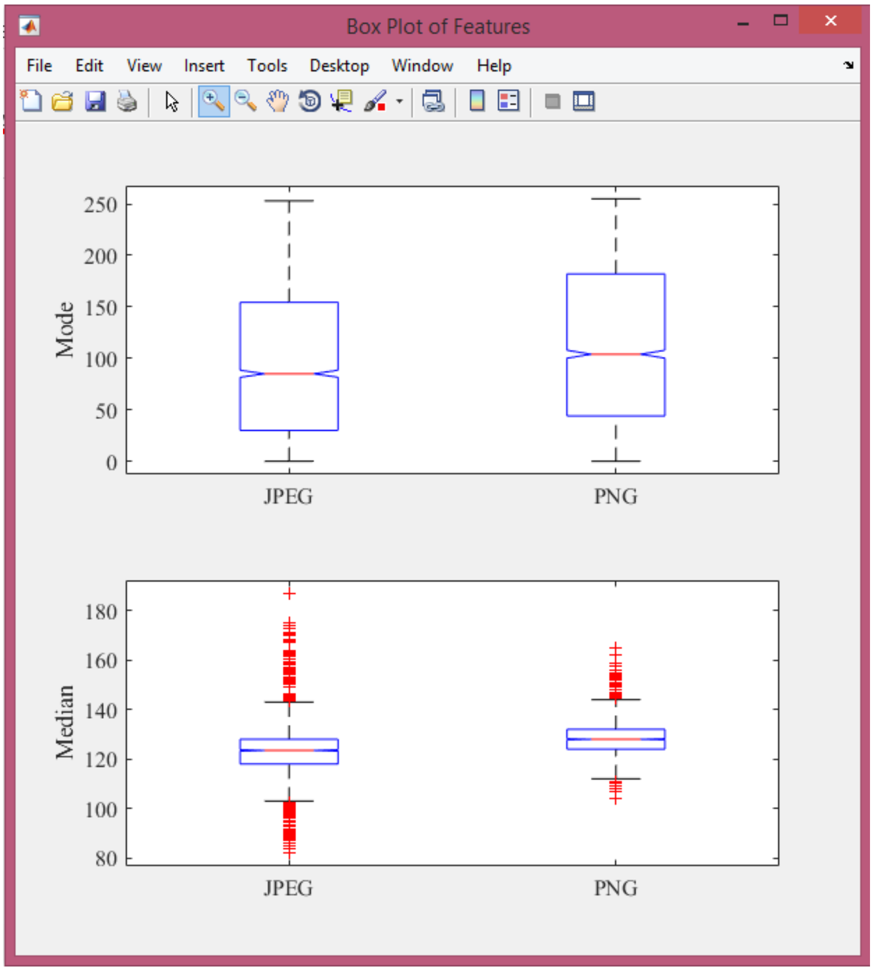


Figure 4-2: An example of box plots.

4-3 Plot Feature Histogram

You can plot the histogram of one or more features for one or more selected class labels. To do so, click on the "Plot Feature Histogram" submenu in the "Visualization" menu. In the next window, select among features. For each chosen feature a separate histogram will be plotted. After confirming the feature label(s), you must choose the desired class label. You can select more than one class. Now you must again confirm the class labels. In the next window, you should set the number of bins for the histogram and the subplot organization. After that, press the OK button. Figure 4-3 shows an example of a histogram.

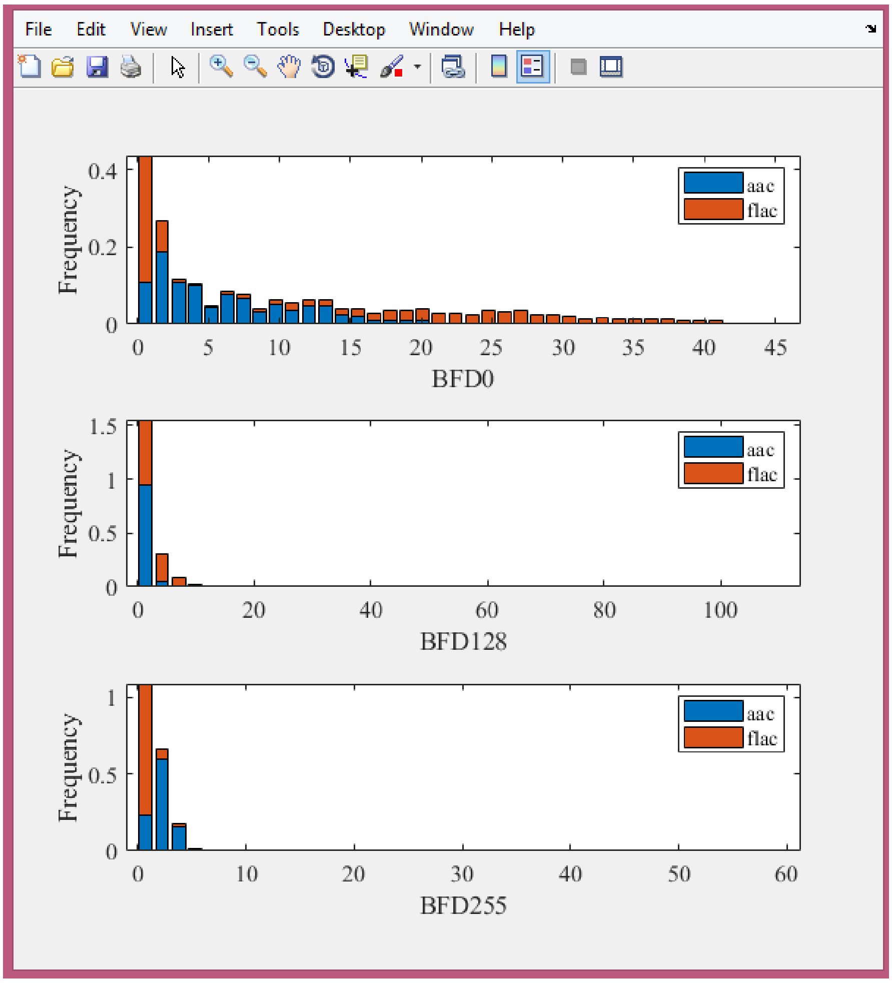


Figure 4-3: An example of a histogram plot.

4-4 Display Samples in Feature Space

You can display samples of multiple class labels in a feature space. You can select two or three features to form a 2-Dimensional (2-D) or 3-Dimensional (3-D) feature space, in which, each sample of the dataset is represented as a data point.

To do so, click on the "Display Samples in Feature Space" submenu in the "Visualization" menu. In the next window select two or three features. Also, you should select some classes. In this process, you must also confirm the feature and class labels. Afterward, you can see the distribution of samples in the selected feature space. In Figure 4-4, an example of a 3-D feature space representation is shown.

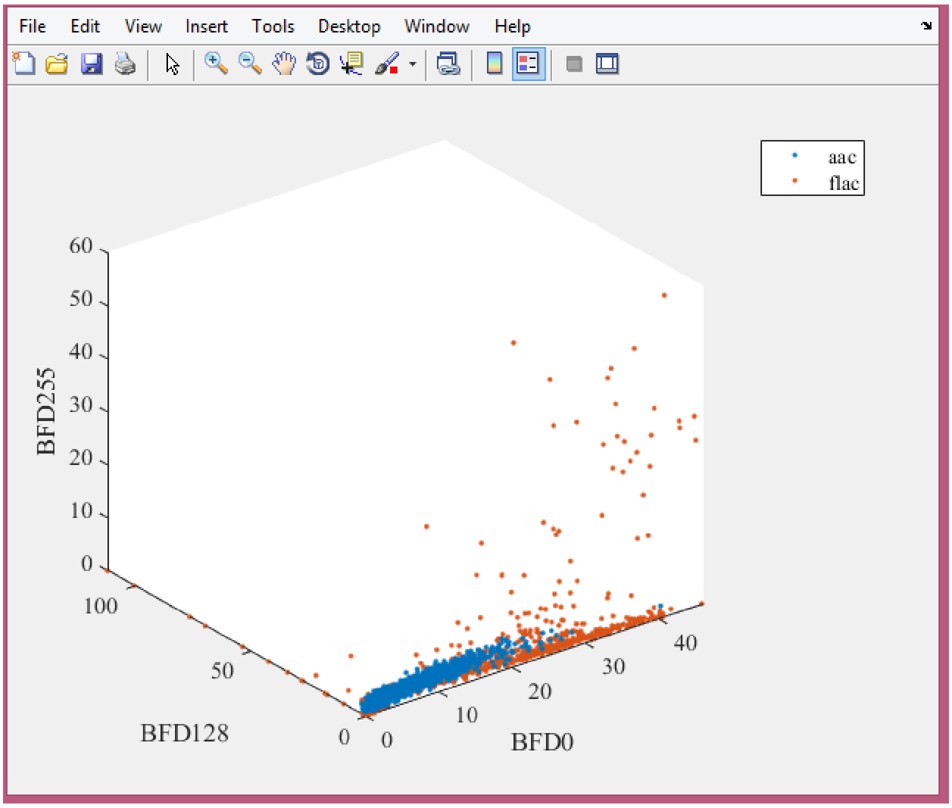


Figure 4-4: An example of displaying samples in a 3-D feature space.

5 Loading Previously Generated Data and Results

If you have saved a dataset, a decision machine, a test result, or a cross-validation result, you can load them again in the software.

5-1 Load Dataset

To load a dataset, click on the “Load Dataset” submenu in the “File” menu. Select the dataset in your device. When the dataset is loaded, you can view class labels and features.

5-2 Load Decision Machine

To load a decision machine, click on the “Load Decision Machine” submenu in the “File” menu. Select the trained machine. When the decision machine is loaded, you can see the validation accuracy.

5-3 Load Test Results

To load a test result, click on the “Load Test Results” submenu in the “File” menu. Select the test results. When it is loaded, you can click on “View Test Results” to see the test results.

5-4 Load Cross-Validation Results

To load a cross-validation result, click on the “Load Cross-Validation Result” submenu in the “File” menu. Select the cross-validation results. When it is loaded, you can click on “View Cross-Validation Results” to see the result of cross-validation. In this case, you can see the confusion matrices in the command window.

Abbreviations

2-D	2-Dimensional
3-D	3-Dimensional
BFD	Byte Frequency Distribution
BRO	Binary Ratio
FNF	False Neighbors Fraction
k-NN	*k*-Nearest Neighbors
LDA	Linear Discriminant Analysis
LE	Lyapunov Exponent
MAD	Mean Absolute Deviation
RMS	Root Mean Squared
RoC	Rate of Change
STD	Standard Deviation
SVM	Support Vector Machine

References

[1] W. C. Calhoun and D. Coles, "Predicting the types of file fragments," *Digital Investigation,* vol. 5, pp. S14-S20, 2008.

[2] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, *et al.*, "Automated mapping of large binary objects using primitive fragment type classification," *digital investigation,* vol. 7, pp. S3-S12, 2010.

[3] A. Swami, J. M. Mendel, and C. L. Nikias, "Higher-order spectral analysis toolbox," *The Mathworks Inc,* vol. 3, pp. 22-26, 1998.

[4] R. F. Erbacher and J. Mulholland, "Identification and localization of data types within large-scale file systems," in *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07),* 2007, pp. 55-70.

[5] S. Hicsonmez, H. T. Sencar, and I. Avcibas, "Audio codec identification from coded and transcoded audios," *Digital Signal Processing,* vol. 23, pp. 1720-1730, 2013.

[6] P. Tripathi, K. P. Raju, and V. R. Chandra, "A Novel Technique for Detection of CVSD Encoded Bit Stream," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 2, pp. 6035-6040, 2014.

[7] J. Goubault-Larrecq and J. Olivain, "Detecting Subverted Cryptographic Protocols by Entropy Checking," *Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-06-13,* 2006.

[8] X. Jin and J. Kim, "Video fragment format classification using optimized discriminative subspace clustering," *Signal Processing: Image Communication,* vol. 40, pp. 26-35, 2016.

[9] X. Jin and J. Kim, "Audio Fragment Identification System," *International Journal of Multimedia and Ubiquitous Engineering,* vol. 9, pp. 307-320, 2014.

[10] F. Kaspar and H. Schuster, "Easily calculable measure for the complexity of spatiotemporal patterns," *Physical Review A,* vol. 36, p. 842, 1987.

[11] T. Xu, M. Xu, Y. Ren, J. Xu, H. Zhang, and N. Zheng, "A File Fragment Classification Method Based on Grayscale Image," *Journal of Computers,* vol. 9, pp. 1863-1870, 2014.

[12] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International journal of computer vision,* vol. 42, pp. 145-175, 2001.

[13] G. Hinto and S. Roweis, "Stochastic Neighbor Embedding," *Advances in Neural Information Processing Systems*, 2003, pp. 857-864.