Content-based File-type Identification Using Cosine Similarity and a Divide-and-Conquer Approach

Irfan Ahmed, Kyung-suk Lhee, Hyunjung Shin[1] and ManPyo Hong

Digital Vaccine and Internet Immune System Lab, Graduate School of Information and Communication,
[1]Department of Industrial and Information Systems Engineering, Ajou University, South Korea

Abstract

Identifying the file type (TXT, EXE, JPEG, etc.) is important for computer security applications such as computer forensics, steganalysis, and antivirus programs. The common approach for this is to use file extensions, magic numbers, or other header information. However, these are susceptible to tampering or corruption; for instance, the file extension can be easily spoofed and the magic numbers can be obfuscated. A more reliable approach may be to analyze the file content instead of using only the tip of the information (metadata). This paper proposes two methods based on the file content. First, we use the cosine distance as a similarity metric when comparing the file content rather than the Mahalanobis distance that is popular and has been used by the other related approaches. The cosine similarity (unlike the Mahalanobis distance) retains the classification accuracy on a small number of highly frequent byte patterns which leads to a smaller model size and faster detection rate. Second, we decompose the identification procedure into two steps by taking the divide and conquer: in the first step, the similar files in terms of byte pattern frequencies are grouped into several clusters. In the next step, the cluster which contains different file types is fed to the neural network in order for finer classification. The experiments showed that the classification followed by clustering leads to higher accuracies.

Keywords

Byte frequency distribution, Cosine similarity, Clustering, File type identification, Mahalanobis distance, Neural network.

1. Introduction

File type (e.g., JPG, DOC, and TXT) identification by computer operating systems is vital in order to manage and process files. For instance, programs usually interpret files as a stream of bytes, and thus it is required to identify file types within the file system. Approaches to this problem vary depending on the operating system, and include the file extension (combining the file type with the name using period), magic number (keeping the file type information in file header), and storing the file type in the file system.

Many security applications require identification of file types in order to work efficiently. For example, email attachment filtering may require blocking types of inbound attachments that may contain malicious contents. Virus scanners may be configured to skip some file extensions; in Norton Antivirus 2008, for example, there is an exclusion option to specify file types that will be skipped by the virus scan [1]. Some applications raise alerts before opening unrecognized (suspicious) file extensions; for instance, Windows Media Player raises an alert when the user tries to open a file with an unrecognized extension. Some steganalysis programs also rely on file type detection; for example, Stegdetect [2], which detects steganographic contents in images, uses the libmagic1 package [3] and magic numbers to determine the file type.

Because the current solutions that rely on metadata information are susceptible to tampering or corruption, they cannot be trusted to securely identify file types. For example, file extensions can be renamed or file headers can be encoded in order to obfuscate the file type information. In the presence of adversaries, analyzing the file content may be more reliable means to identify file types. The file contents consist of bytes and 1 byte has 256 unique values, i.e. 0–255; these are denoted as byte patterns in this paper. The total number of occurrences of each byte pattern in a file is called the byte frequency. Solutions such as *fileprint* [4] and *fingerprint* [5] that analyze the file content usually compute the byte frequency vector and learn the distinguishable byte patterns of file types. But such schemes fail to achieve high classification accuracy.

This paper proposes two methods to improve the per-

formance of content-based file type identification. The first one is to use the cosine similarity for byte frequency comparison. The cosine similarity ensures classification accuracy when using only certain percentage of high-frequency byte patterns; this results in a smaller model size and faster detection rate. Using high-frequency byte patterns can effectively eliminate nonrepresentative patterns that rarely occur in file types. Thus, it can improve the classification accuracy as well. We compare the cosine similarity with the Mahalanobis distance (popular metric used by many schemes such as Fileprint [4], PAYL [6-8]).

The second one is to use clustering ahead of file type identification: the divide-and-conquer approach. We group similar files in terms of their byte pattern frequencies irrespective of their file types. The grouping is based on the clustering technique. Then we perform classification of the resulting clusters that are mixed up with different types of files. In this paper, we use a neural network as the classifier. We show that the divide-and-conquer approach improves the classification accuracy over the classification algorithm used without clustering.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 describes the two proposed methods. Section 4 discusses the experimental details and the empirical results of the proposed methods, followed by the conclusion in section 5.

2. Related Works

This section introduces previous works on file type identification, some of which are commonly used in operating systems where the file type information is explicitly written in the file header or name. Other techniques analyze the file content in order to identify the file type.

2.1 Using File Headers

The most common and simplest means to identify the file type is to examine the file's extension [9], but this can be easily spoofed by users with malicious intent. Novices can also unintentionally change the file extension while renaming the file. Malwares can also be easily hidden by using file extensions that are skipped by virus scanners.

Another method to identify the file type is to examine the magic numbers in the file [10]. For example, GIF files begin with an ASCII representation of GIF87a or GIF89a depending on the standard. ZIP files always have the magic number PK at the beginning of the file; however, because only binary files have magic numbers, this is not applicable to text files. As with the file extension, it can also be easily spoofed. For instance, malcodes can be hidden using code obfuscation and alphanumeric encoding [11-15].

Li *et al*. [16] categorize a file header fragment based on its contents by using the support vector classifier algorithm. They use an enhanced string kernel (ESK) to discover byte-level patterns in the file header fragment and use an extended suffix array data structure to efficiently store and manipulate the feature map. They use five file types, i.e., MS Word, MS Excel, JPG, C++, and PDF. The classification accuracy of each file type varies from 61% to 100% depending on the minimum sequence length of the suffix within the sliding window.

2.2 Using File Contents

This section presents variety of content-based file type identification schemes. Since the first two schemes are closet to our divide-and-conquer approach, they are also applied to our dataset in order to compare their classification accuracies with our approach.

Li *et al*. [4] identify file types using *n*-gram analysis. They calculate 1-gram frequency distribution of files and build three different models of each file type: single-centroid (one model of each file type), multicentroid (multiple models of each file type), and exemplar files (the set of files of each file type) as centroid. They refer to these as the 'fileprint'. In single-centroid and multi-centroid models, they calculate the mean and standard deviation of the 1-gram frequency distribution of files, and use the Mahalanobis distance to compare these models with the 1-gram distribution of the given file and find the closest model. In the exemplar file model, they compare the 1-gram distribution of the exemplar file with that of the given file (no variance is computed), and the Manhattan distance is used instead of the Mahalanobis distance. Their solution cannot identify files that have similar byte frequency distributions such as MS Office file formats (e.g., Word and Excel), but it treats them as a single group or abstract file type.

Harris [14] used an artificial neural network to identify file types. The file is divided into blocks of 512 bytes and only the first 10 blocks are used for file type identification. Two features are extracted from each block, i.e., raw filtering and character code frequency. Raw filtering takes each byte as an input for one neuron of the neural network. However, the character code frequency counts the number of occurrences of each character code in the four blocks and takes the frequency of the character as an input for the neurons. It is assumed that raw filtering is useful for files for which byte patterns occur at regular intervals; however, the character code

frequency is useful for files which have irregular byte pattern occurrences. He only used image files, i.e., JPG, PNG, TIF, GIF, and BMP, as a sample set and reported a detection rate ranging from 1% (GIF) to 50% (TIF) for raw filtering and from 0% (GIF) to 60% (TIF) for character code frequency.

McDaniel and Heydari [5] introduced three algorithms to identify file types by analyzing file content. In the byte frequency analysis (BFA) algorithm, they calculate the byte frequency distributions of different files and generate the 'fingerprint' of each file type by averaging the respective byte frequency distributions. They also calculate the correlation strength as another characterizing factor. They take the difference between given byte values in different files. As the difference decreases, the correlation strength approaches to 1 and vice versa. They use the byte frequency cross-correlation algorithm to find the correlation between all the byte pairs. They calculate the average frequency between all the byte pairs and the correlation strength, in a similar manner to the BFA algorithm. In the file header/trailer algorithm, the file headers/trailers are byte patterns that appear at a fixed location at the beginning/end of the file. They maintain an array of 256 entries for each location and the array entry corresponding to the byte value is initialized with the correlation strength of 1. They construct the fingerprint by averaging the correlation strength of each file and use these algorithms to compare the file with all the generated fingerprints and identify its file type.

Martin and Nahid [17,18] proposed the 'Oscar' method for identifying types of file fragments. They build single-centroid fileprints [4] but use the quadratic distance metric and the 1-norm as distance metric to compare the centroid with the byte frequency distribution of the file. Although Oscar identifies any file type, it is optimized for JPEG files. They introduce the rate of change (RoC) as the new metric which is defined as the difference between two consecutive byte values considering the ordering information of bytes. They reported 99.2% detection rate of JPEG files, which is nearly perfect. Such detection rate is possible for JPEG files, because the JPEG format uses 0xFF as an escape character for all metadata tags, and an extra 0x00 is added after every 0xFF byte in the body of the file in order to avoid ambiguity. Thus, a regular, unique, and exploitable pattern, i.e., 0xFF00 does exist in JPEG file, which has a very high RoC. However, the technique does not improve the classification accuracy of file types other than JPEG.

Amirani *et al*. [19] used the hierarchical feature extraction method to more effectively exploit the byte frequency distributions of files for file type identification. They believed that the multiplicity of features reduces the speed and accuracy for file type identification. Thus, they utilize principal component analysis and an auto-associative neural network to reduce the 256 features of byte patterns to certain smaller number (where the detection error is negligible). After feature extraction, they use three-layer multilayer perceptron (MLP) for detecting the file types. They used the DOC, PDF, EXE, JPG, HTM, and GIF file types for the experiments, each consisting of 30 test data points, and they reported an accuracy of 98.33%. However, the choice of file types used in their experiments may have been biased. That is, although their experimental results show high accuracy in distinguishing different characteristics of file types such as HTML and EXE, they didn't try to distinguish similar characteristics of file types, e.g., TXT and HTML that only contain ASCII printable characters and can be confused during detection.

Moody and Erbacher [20] aimed to identify the embedded data types in files. Files of type DOC and XLS can import a variety of objects (e.g., image files); thus these whole files do not represent a single type. They vary the window sizing to process a small chunk of file data. They gathered a variety of statistical characteristics of byte values in each window, including the average, standard deviation, and kurtosis and used them to identify the type of chunk. They reported 74.2% detection accuracy, which includes the high detection rate of text-type files, i.e., CSV (100%), HTML (100%), and TXT (80%).

Hall and Davis [21] determine file types using entropy and compressibility measurements. They use sliding window and calculate the entropy and compressibility value of each window. The values of each file type are used to obtain the average and standard deviation, and these are compared with the test file by using point-by-point delta and Pearson's rank order correlation to identify file types.

Veenman [22] extracts three features from the file content: (1) the byte frequency distribution, (2) the entropy derived from the byte frequency distributions of files, and (3) the algorithmic or Kolmogorov complexity that exploits the substring order [23]. The Fisher linear discriminant is applied to these features in order to identify the file type.

Calhoun and Coles [24] extended Veenman's work by building classification models (based on statistics such as the ASCII frequency, entropy, etc.) and applying the linear discriminant to identify file types. They also argued that files of the same type probably have longer common substrings than those of different types. Our approach also uses the byte frequency distribution as feature and linear discriminant analysis for clas-

sification. However, the main difference between our method and Veenman's is the means by which we build the classification model for each file type. Veenman computes a single discriminant function using all the sample files for each file type. However, our approach combines similar byte frequency files into groups by using clustering, irrespective of their file types, and computes the linear discriminant function for each file type in each group. Hence, multiple functions can be computed for each file type.

This paper is an extension of our preliminary work [25]. In this paper, we extend our work by using a neural network to improve the classification accuracy of our proposed method. We also present its comparison with Harris's approach (also based on a neural network). The comparison substantiates our claim that file grouping irrespective of the file type improves the classification accuracy for a given classification algorithm.

3. Proposed Methods

3.1 Cosine Similarity and High-Frequency Byte Patterns

The first approach we suggest in this paper is to use the cosine similarity for vector comparison. It retains the classification accuracy when only using certain percentage of high-frequency byte patterns. Thus, it results in a smaller model size and a faster detection rate. Moreover, we conjecture that byte patterns that rarely occur in a file type do not represent its normal patterns. If they are present in the representative model, they can degrade the classification accuracy. In that case, using a subset of high-frequency byte patterns is an effective means to eliminate them and improve the classification accuracy as well.

3.1.1 Cosine Similarity

If x is a byte frequency vector of a test file and $\overline{y}$ is the representative model vector of the file type (obtained by averaging the byte frequency distributions of the file type's sample files), then the cosine similarity is defined as

$$d_c = cos(x, \overline{y}) = \frac{x \cdot \overline{y}}{|x||\overline{y}|} \tag{1}$$

where the dot '.' indicates the vector dot product, $x \cdot \overline{y} = \sum_{k=1}^{n} x_k \overline{y}_k$ and $|x| = \sqrt{\sum_{k=1}^{n} x_k^2} = \sqrt{x \cdot x}$ is the length of the vector. The cosine similarity is in the range [0,1]. When the value is 1, the angle between x and $\overline{y}$ is 0; thus x and $\overline{y}$ are the same except for the magnitude. When it is 0, the angle is 90°, which means that they are dissimilar.

To prove the superiority of the cosine similarity, we compare it with the well-known measure, based on the Mahalanobis distance [26]. A simplified Mahalanobis distance [4] is defined as follows:

$$d_m = d(x, \overline{y}) = \sum_{i=1}^{n-1} \frac{(|x_i - \overline{y}_i|)}{\overline{\delta}_i} \tag{2}$$

where $\overline{y}$ and $\overline{\delta}$ are the respective mean value and standard deviation of the byte frequency distributions of the sample files used to build the representative file type.

The cosine similarity is different from the simplified Mahalanobis distance, in that it puts more emphasis on inherent characteristics of file types (angle between two types of files) than magnitude (i.e., file size). Therefore, normalization for sizes of files is not necessary unlike the Mahalanobis distance.

3.1.2 Selecting a Subset of High-frequency Byte Patterns

We obtain the number of subsets of file type's high-frequency byte patterns via the following three steps: (1) we average the byte frequency distributions of each file type's files in order to ensure that the usual frequency of byte patterns is present; (2) the byte patterns are sorted in a descending order with respect to their frequencies, and (3) starting from the head of the sorted list, we extract the number of subsets containing different percentages of high-frequency byte patterns where $S_i \subseteq S_{i+1}$ if $S_i < S_{i+1}$ and here S_i represents the subset i of a file type. For the same percentage of high-frequency byte patterns, it is found that different file types may have different subsets of byte patterns.

3.2 Divide and Conquer

The other approach proposed in this paper consists of two steps. It first groups the files using clustering and then later (at second step) applies classification to the group which is heterogeneous in terms of file types of the group members. We group files without considering their file types.

The motivation behind this approach is that the multi-centroid modeling technique (by Li *et al*. [4]) groups files with respect to their file types and builds multiple models for each file type. Thus, similar models of different file types can be built if files of different types can have similar byte frequency distributions, and therefore, file types can be confused; this is an inherent problem with their scheme (refer to section 4.2.2 for details).

3.2.1 Clustering

Clustering is an exploratory statistical procedure to

naturally group data into different clusters. At this stage, we do not consider the file type. Thus, clustering divides files of varying types (in the sample space) into a few clusters, each of which contain files with similar byte frequency patterns. Each cluster can have files of one or many different types. For instance, Figure 1(b) illustrates clustering where DOC, TXT, and ASP file types with similar byte frequency distributions are grouped together into three clusters.

3.2.2 *Classification*

After clustering, a classification algorithm is applied to a cluster that has files of multiple types in order to identify each type. For instance, Figure 1(c) illustrates that the classification algorithm draws margins among file types to separate them for clusters having files of multiple types.

This paper uses a neural network (NN) as a classification algorithm, which is popular and well-known in pattern recognition [27]. We use a MLP neural network that has input, output, and hidden nodes (refer to Figure 2). Hidden nodes are intermediate nodes that are not input or output nodes. There are 256 input nodes, each representing one unique byte pattern, for which its frequency is passed as an input value. There is only a single hidden layer in the neural network. The n numbers of hidden nodes are obtained on a hit-and-trial basis. The number of output nodes is the m types of files in a cluster. For instance, the NN has two output nodes for the cluster having ASP and TXT file types in Figure 1(c).

3.2.3 *Type Identification Process of a File*

Figure 3 describes the file type identification of the proposed system when a new file of an unknown type enters into the system. At first, the relative byte frequencies of the files are computed; then, the file is assigned to the cluster with the most similar byte pattern frequencies. If the assigned cluster represents only one file type, the same type is assigned to the file; otherwise a classification method is used to identify the exact file type. Misclassification can occur during both steps, i.e. clustering and classification. During clustering, if a test file's type does not match any type of the assigned cluster, it is said to have been misclassified. For example, if file f of type X is assigned to cluster Y but this cluster does not have type X, then file f is considered to have been misclassified.

If not misclassified but assigned to the cluster composed of multiple file types, it is further fed to the classifier.

4. Experiments

4.1 Dataset

We used 10 file types (JPG, HTML, GIF, EXE, MP3, PDF, TXT, DOC, XLS, and ASP) each of having 200 files (refer to Table 1), and hence, 2000 files in all. These file types were chosen because of their popularity and the fact that they cover a broad range of file contents including binary, text, and compressed files. Fifty percent of the files were used for the training dataset to build representative models of file types and the remaining 50% were used for the test dataset to test the classification accuracy of the identification schemes.

The file type represents how the data is encoded in a file. However, the implementation of the encoding schemes

Table 1: Details of the dataset used for experiments

File type	Quantity	Average size (Kilo bytes)	Minimum size (Bytes)	Maximum size (Kilo bytes)
ASP	200	3.52	49	37
DOC	200	306.44	219	7,255
EXE	200	522.71	882	35,777
GIF	200	3.24	64	762
HTML	200	11.59	117	573
JPG	200	1,208.27	21,815	7,267
MP3	200	6,027.76	235	30,243
PDF	200	1,501.12	219	32,592
TXT	200	269.03	16	69,677
XLS	200	215.98	80	9,892

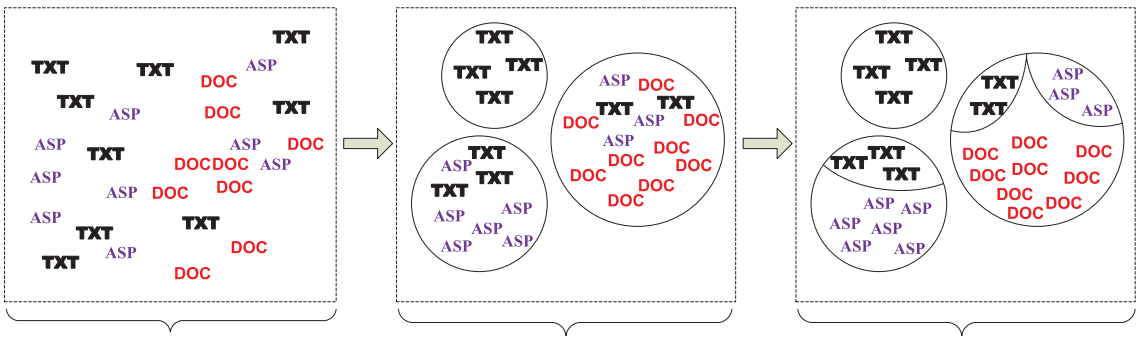


Figure 1: Schematic illustration of file type classification followed by clustering.

might slightly differ across software. Thus, we collected sample files from different sources in order to ensure that a file type's sample files were not generated by a single source. Thus, executable files were mostly obtained from the *bin* and *system32* folders from the Linux and Windows XP operating systems, respectively. Moreover, other files were collected from the internet using a general search on *Google*. For example, we search for .txt files using option *filetype:txt*. Image files such as GIF and JPG were also obtained from photo-sharing websites such as *Picassa* of *Google*, *Flickr*, etc. MP3 files were collected from different random sources mostly from publically available FTP servers for movies and personal computers. In short, such a random collection of files can be considered an unbiased and representative sample of the given file types.

4.2 Results on the Cosine Similarity and High-Frequency Byte Patterns

4.2.1 Experimental Settings

We built multiple representative models of each file type using a number of different subsets of the file type's high-frequency byte patterns. The two modeling techniques, single-centroid and multicentroid, are used for experimentation.

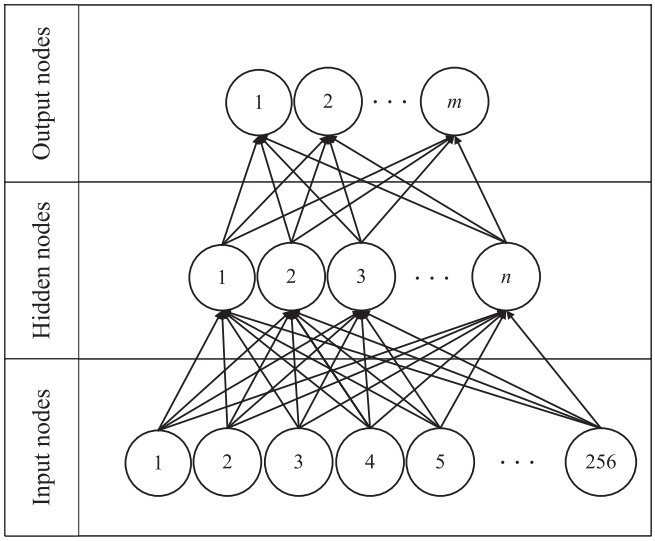


Figure 2: Structure of the MLP neural network.

For the single-centroid modeling technique, we first computed the byte frequency distribution of each file and divided each byte pattern frequency by the file size; then we considered each byte pattern's relative frequency as a variable, and computed its mean and standard deviation for each file type. For the multicentroid model, we use the *k*-means algorithm to group similar byte frequency files with respect to their file types. We chose a random value of *k* (3) as Li *et al*. [4] have reported similar results for different values of *k*. Thus, *k* multiple models were built for each file type. Each model contained a mean and standard deviation that were computed in a similar manner to the single-centroid model. We adopted Li *et al.'s* work because they use the Mahalanobis distance as a comparison metric and their single-centroid model is also used by other methods [17-22]. Moreover, their multicentroid model is comparable to our divide-and-conquer approach.

We use the simplified Mahalanobis distance formula and the cosine similarity to compare the test file's byte frequency distributions with the models. The aim is to find the classification accuracy of both comparison metrics for different percentages of high-frequency byte patterns (refer to the subsequent section). The accuracy (%) is calculated using the following equation:

$$\text{Accuracy}(\%) = \frac{(\text{Total no. of files - misclassified files})}{\text{Total no. of files}} \quad (3)$$

4.2.2 Analysis of Empirical Results

Figure 4 shows the average classification accuracy of the Mahalanobis distance and the cosine similarity (refer to the appendix for the individual file-type comparison). We notice that the cosine similarity retains its clas-

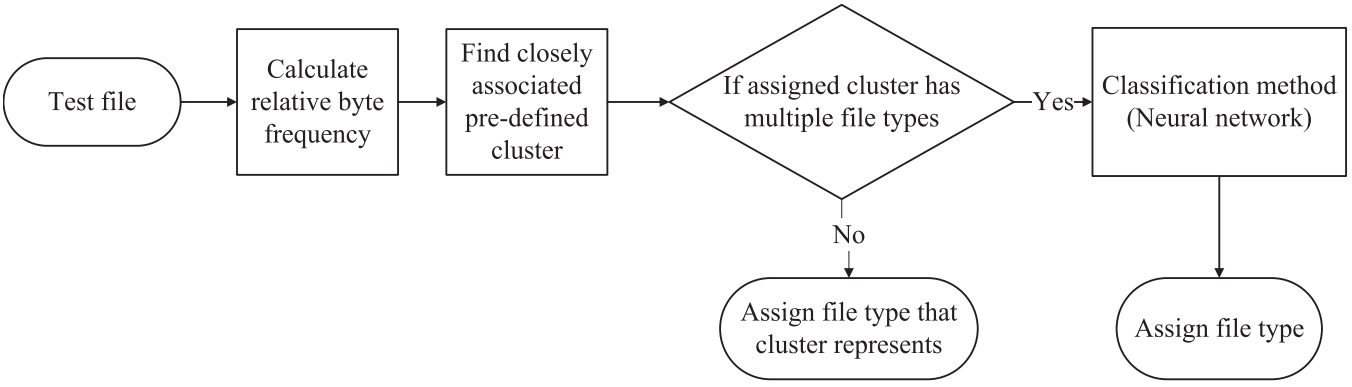


Figure 3: Identifying a test file using the divide-and-conquer approach.

sification accuracy when using different percentages (10–100%) of byte patterns (high-frequency). The same results are obtained for most of the given file types; for the cosine similarity, the classification accuracy remains the same or improves as the number of byte patterns in the model decreases. However, the classification accuracy is degraded for the Mahalanobis distance.

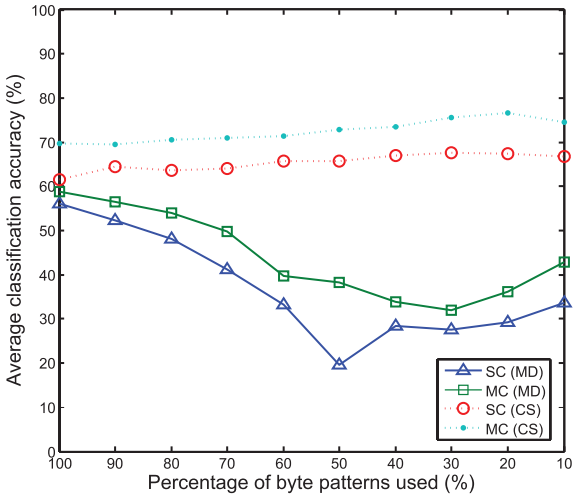


Figure 4: Average classification accuracy of single-centroid (SC) and multicentroid (MC) models for the cosine similarity (CS) and Mahalanobis distance (MD) using 10 file types.

It is also noticed in Figure 4 that the multicentroid model shows better accuracy than the single-centroid one. However, the average classification accuracy for Li *et al.'s* modeling technique is less than 80%, which is relatively low and may be unsatisfactory for many applications. In order to identify the cause of this, we analyzed the byte frequency distribution of the file types in the dataset. The following two difficulties were found.

1. Byte frequency distributions of different file types can be similar. That is, they can have similar graphical representations of byte frequency distributions (see Figure 5 (a) and (b)). For instance, the text files such as ASP and HTML show this characteristic as they contain hypertext markup language.

2. Byte frequency distributions from a same file type can be different. That is, the files of the same type can have different graphical representations of byte-frequency distributions (see Figure 5 (c) and (d)). For instance, compound files such as DOC and XLS files can embed different types of files (such as image files, etc.) in a single file.

A robust solution for file-type identification should cope with these difficulties. However, the single- and multi-

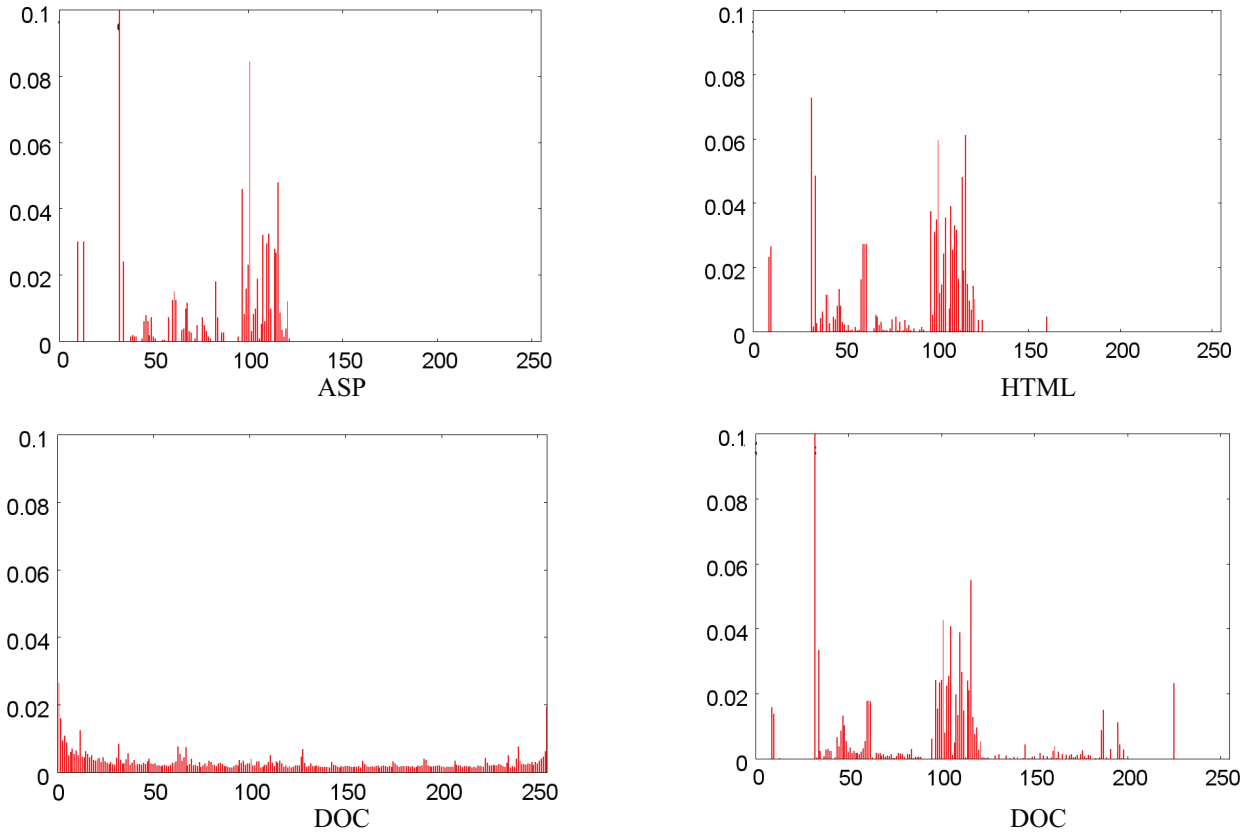


Figure 5: (a) and (b) show that the ASP and HTML file types can have similar byte frequency patterns; (c) and (d) show that two DOC files can have different byte frequency patterns (the *x*- and *y*-axes represent the byte patterns and the relative byte frequency, respectively).

centroid modeling techniques of Li *et al*. do not consider them and thus, they fail to achieve high classification accuracy. The single-centroid modeling technique averages the byte frequency distribution of a file type's files and builds a single representative model for each file type. Thus, it cannot yield an accurate representative model if the file type has more than one normal byte frequency distribution (see Figure 6). The multicentroid modeling technique groups files with respect to their file types and builds multiple models for each file type. Thus, it can build similar models of different file types if files of different types can have similar byte frequency distributions,

and therefore it can confuse file types (see Figure 7). Table 2 shows the average percentage of file types that are confused with other types, as noted during the experiments.

Figure 8 shows the cumulative improvement in elapsed time as the percentage of byte patterns is reduced. It is noticed that the use of only 10% of byte patterns can reduce the elapsed time by approximately 11%. Our experiment considers each byte pattern as a variable (i.e., 1-gram analysis). Since the size of the model using a 1-gram frequency distribution is small (i.e., limited to 256 patterns), there is little scope for improvement in memory space and time. However, models using higher *n*-grams [28,29] (i.e., a variable consisting of a sequence of multiple bytes) are much bigger, since the number of distinct patterns grows exponentially. Therefore, the improvement in model size and computation time would be much greater if we used higher *n*-grams (but these are beyond the scope of this paper).

In short, the empirical results suggest that we can use a small percentage of high-frequency byte patterns with the cosine similarity, and this reduces the model size and improves the computation time for the cosine similarity.

4.3 Results on the Divide-and-Conquer Approach

4.3.1 Experimental Settings

We use Ward's clustering method [30,31], to group similar byte frequency files irrespective of their file types. It forms clusters by minimizing the total within-cluster sum

Table 2: Percentage of file types confused with other file types during detection

Actual file type	SC using CS	MC using CS	SC using MD	MC Using MD
ASP	HTML (13.1%)	TXT (90.7%)	HTML (11.9%) TXT (2.1%)	GIF (2.9%) TXT (68%)
DOC	XLS (80.3%)	GIF (10.3%) TXT (27.9%) XLS (31.4%)	EXE (15.8%) XLS (22.7%)	ASP (2.1%) GIF (24%) TXT (19.1%) XLS (16%)
EXE	DOC (18.5%) XLS (38.9%)	DOC (7.2%) TXT (22.4%) XLS (25.5%)	DOC (11.1%)	DOC (21.9%) GIF (11.6%) TXT (8.6%) XLS (11.5%)
GIF	DOC (9.6%) EXE (4.2%) JPG (3.1%)	TXT (17.5%) XLS (8.1%)	DOC (18.5%) EXE (3.1%) MP3 (4.5%)	DOC (8.8%) TXT (7.4%)
HTML	ASP (11%) TXT (4%)	TXT (81.6%)	ASP (4.2%)	TXT (59.3%)
JPG	GIF (39.3%) MP3 (2.5%)	DOC (8.5%) GIF (74.8%) XLS (15.2%)	GIF (19%) MP3 (45.8%)	DOC (7.1%) GIF (84.7%)
MP3	GIF (3.1%) JPG (4.9%)	DOC (9.9%) GIF (68.7%) XLS (17.6%)	JPG (1.3%)	DOC (13.5%) GIF (55.5%)
PDF	PDF (5.9%) JPG (12.2%)	DOC (7.3%) GIF (32.4%) TXT (13.8%) XLS (8.6%)	JPG (2.6%) MP3 (7.5%)	DOC (27.4%) GIF (27.2%) TXT (8.4%)
TXT	ASP (34%) HTML (17%) PDF (3.8%)	XLS (2.5%)	ASP (14%) HTML (24.9%) PDF (4.9%) XLS (4.3%)	GIF (4.5%) DOC (2.8%)
XLS	DOC (5.1%)	DOC (3.9%) GIF (4.1%) TXT (23.3%)	DOC (26%) EXE (7.2%) HTML (4%)	DOC (8.1%) GIF (14%) TXT (20.8%)

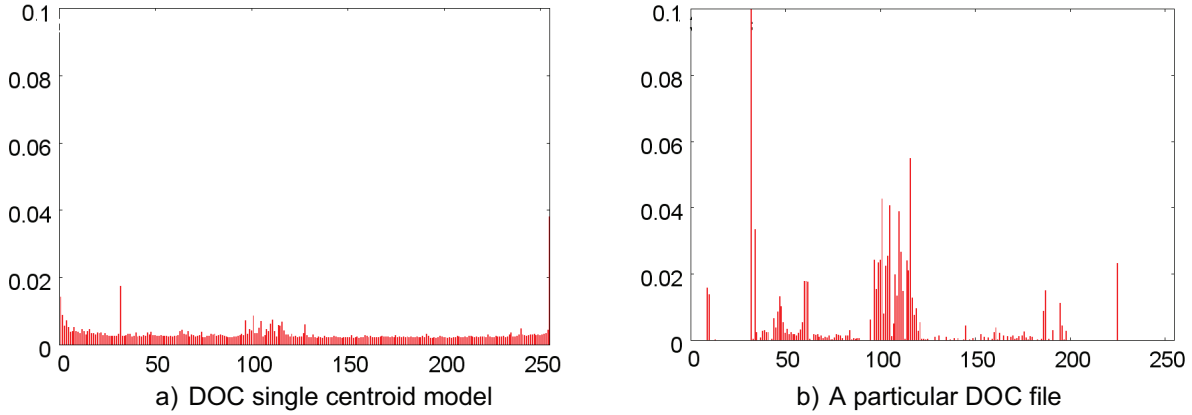


Figure 6: Inaccurate single-centroid model of the DOC file type (the *x*- and *y*-axes represent the byte patterns and the relative byte frequency, respectively).

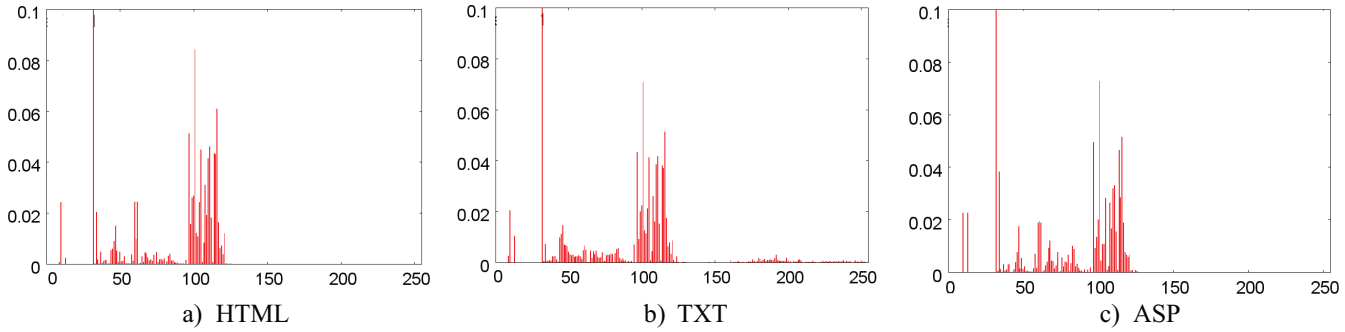


Figure 7: Similar multicentroid models of different file types (the *x*- and *y*-axes represent the byte patterns and the relative byte frequency, respectively).

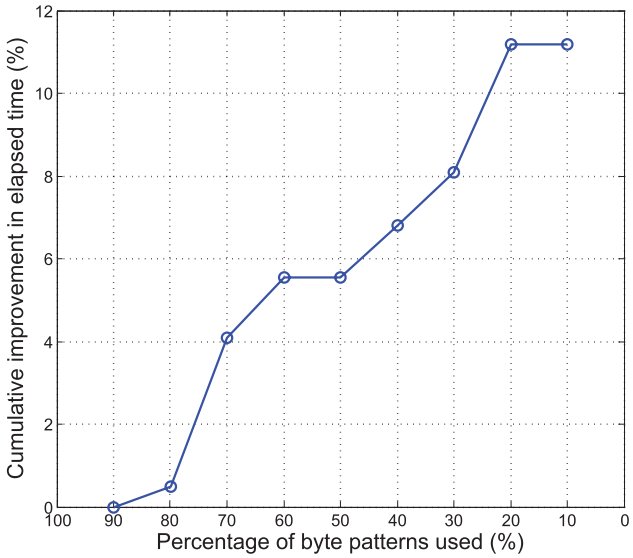


Figure 8: Cumulative improvement in elapsed file-type detection time for processing 1500 files (size: 419 MB).

of squares. For instance, if XY is the cluster obtained by combining clusters X and Y, the sum of within-cluster distances (W_{XY}) are

$$W_{XY} = \sum_{i=1}^{n_{XY}} (y_i - \overline{y}_{XY})'(y_i - \overline{y}_{XY}) \quad (4)$$

where y_i are data points, $\overline{y}_{XY} = \frac{(n_X \overline{y}_X + n_Y \overline{y}_Y)}{(n_X + n_Y)}$, and n_X, n_Y, and $n_{XY} = n_X + n_Y$ are the number of data points in X, Y, and XY, respectively.

The NN is configured as follows. The NN has a structure of 256 input nodes, 6 hidden nodes, and the same number of output nodes as that of the file types in the cluster. The number of hidden nodes is set to 6 as there is no further improvement in the classification accuracy with more nodes. It is expected that there is a point beyond which additional hidden nodes do not provide further benefits; this phenomenon is mentioned by Liang *et al*. [32].

Dua *et al*. [33] mentioned that a linear activation function effectively negates the benefits of a tri-layer network. Thus the activation function is set to a hyperbolic tangent and the learning rate is set to 0.1.

The SAS e-miner [34] is used for the experiments.

4.3.2 *Analysis of Empirical Results*

Figure 9 illustrates the grouping of different file types and the respective percentages. We group files in 3, 6, and 8 clusters. Figure 9 shows that files of different types with similar byte frequency distributions are grouped together in one cluster. However, the file types with dissimilar byte frequency distributions lie in multiple clusters. We also noted some outlier clusters that were not used in the type identification process; we considered these to be outliers because each consists of only one TXT file.

The divide-and-conquer approach achieved the best accuracy when using 6 clusters (refer to Figure 9). Thus, we compare it with Harris's work based on the NN [14] and Li *et al*.'s multicentroid modeling technique [4]. By comparing our clustering approach with Harris's work, we can observe the impact of clustering if it is applied before the NN. Moreover, Li *et al*.'s multicentroid model is closest to our work because they also grouped files before building representative models. However, the file grouping method depends on the file types.

Figure 10 shows that clustering before applying the NN improves the classification accuracy of the NN for given file types JPG, MP3, PDF, XLS, and ASP by 7.6%, 20.0%, 7.0%, 10.2%, and 9.14%, respectively.

It is noticed that (1) these file types often lie in one cluster, which implies that the byte frequency distributions of files of the same type are similar. Thus, there is a high probability that files of such types were assigned to the right cluster during the detection phase. (2) Clustering reduces the number of classes (file types) because the NN is trained for each cluster independently, and a cluster

often contains fewer file types than the total number of given file types.

Moreover, since all the file type's sample files are grouped in one cluster, the NN can better generalize patterns for classification for fewer file types, thus improving the classification accuracy.

On the other hand, the remaining file types (i.e., EXE, HTML, GIF, DOC, and TXT) usually lie in multiple clusters, implying that these file types have multiple normal byte frequency distributions. If the sample space used for learning does not contain all the normal byte frequency distributions of a file type, and new distribution patterns of a file type appear during detection process, it is highly likely that the file will be assigned to the wrong cluster.

Moreover, since the files of such file types are divided into different clusters, the NN cannot effectively generalize patterns to accommodate new samples. Therefore, considering these two cases, misclassification occurs at both levels, i.e., clustering and classification; this apparently reduces the classification accuracy.

Figure 10 also shows the comparison with the multi-centroid modeling technique. It is found that both the divide-and-conquer approach and the Harris one achieve higher accuracy for all file types. The weaknesses in the multicentroid models were discussed in section 4.2.2.

Table 3 presents the average classification accuracy of the methods. The results suggest that our method improves the accuracy by 2.00% and 53.4% over Harris's and Li *et al.'s* techniques, respectively.

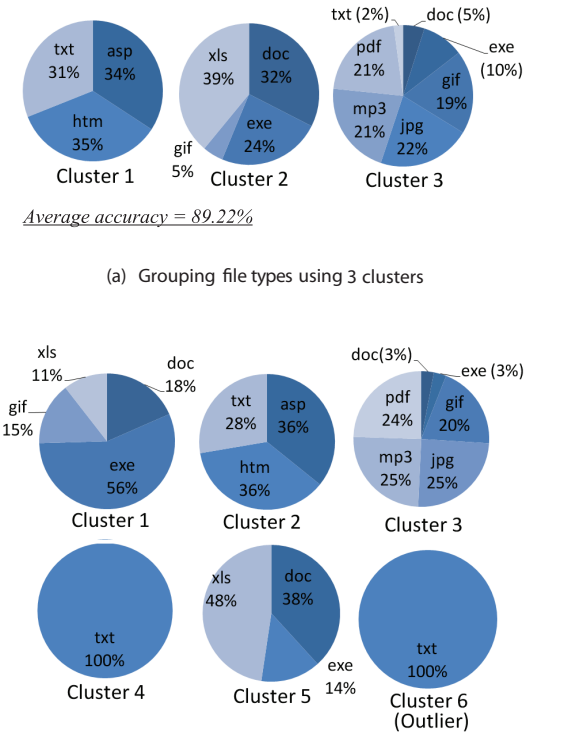


Figure 9: Grouping of file types for different numbers of clusters.

5. Conclusion

This paper presented two new methods to identify file types based on file contents. The first method used the cosine similarity as a distance metric and a certain percentage of high-frequency byte patterns. The second method used a divide-and-conquer approach to identify file types; it involved clustering and classification (using a NN).

Table 3: Average classification accuracy comparison between our proposed method and related works

Methods	Average classification accuracy
Proposed (multi-groups with clustering + NN)	90.19%
Harris's [14] (NN only)	88.5%
Li *et al's* [4] (multi-groups with multi-centroid)	58.70%

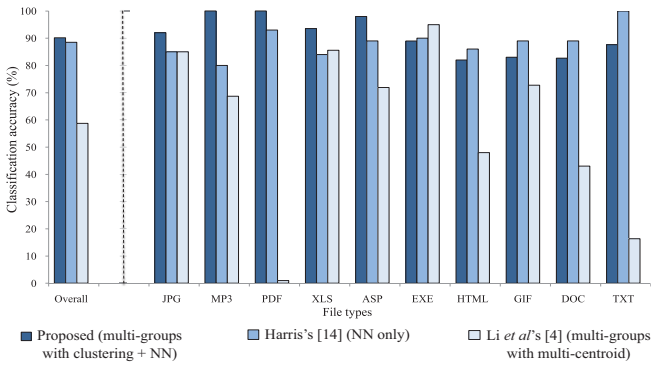


Figure 10: File-type comparison between the divide-and-conquer approach and related works.

Our conclusions about the cosine similarity (CS) are summarized as follows:

- The CS is inherently more accurate than the Mahalanobis distance (MD). For most file types, the CS gives higher accuracy than the MD. For example, on average, the accuracy of the CS is 18.62% higher than that of the MD (when multicentroid models are used).
- Moreover, when we reduce the number of byte patterns, the CS retains almost the same accuracy whereas the MD loses the accuracy. For example, the accuracy of the CS is almost the same even if we use only 10% of byte patterns.
- Since the CS can use less number of byte patterns without compromising accuracy, it can have a smaller model size and a faster detection rate. For example, using 10% of byte patterns, the CS improves the detection speed by approximately 11%.
- The improvement in the model size and the computation time would be much greater if we used a higher *n*-gram.
- Our conclusions about the divide-and-conquer approach are summarized as follows:
- The divide-and-conquer approach can improve the classification accuracy over the classification algorithm used without clustering. For example, for five file types the proposed approach improved the accuracy (ranging from 7% to 20%) compared to the original NN. In our experiment we noticed that such a file type was found in a single cluster.
- However, the accuracy of the proposed approach was lower for the other five file types. In our experiment such a file type was found in multiple clusters; this is probably because such a file type has a large number of byte frequency distributions, some of which were therefore not discovered in the learning phase. We conjecture that the proposed approach sometimes shows lower accuracy than the original NN because of the occasional poor accuracy of the clustering algorithm.

We suggest two ways to further improve the divide-and-conquer approach:

- We can first apply our divide-and-conquer algorithm and, if the identified type is turned out to be the one of those less accurate types, then we can apply the original NN again. Although this will increase the detection time, in terms of the accuracy we can achieve the benefit of both our approach and the original NN.
- We can use a more accurate clustering algorithm, which will enhance the overall accuracy of the divide-and-conquer approach.

In summary, we can use the CS or divide-and-conquer approach in applications where they are best suitable for. The CS can be as accurate as the MD, but uses much smaller model size and is much faster. Therefore, the CS may be useful where reasonable accuracy is required but the fast detection speed is essential. For example, it would be suitable for online processing such as packet filtering requiring payload scanning [8]. The divide-and-conquer approach takes longer time than the CS but gives a better accuracy. Therefore, it may be suitable where the detection accuracy is the prime objective or for offline applications (where the time is not critical).

In our experiment, the text-based types (ASP, TXT, and HTML) seem to be difficult for the divide-and-conquer approach. In the future, we will build a more robust solution that can effectively distinguish confusing file types such as ASP, TXT, and HTML. To this end, it might be helpful to understand the file formats, i.e., how information is encoded in different types of files.

6. Acknowledgments

The authors are grateful to the anonymous reviewers for their valuable feedback which has significantly improved the quality and the presentation of the paper.

This research is supported by the Ubiquitous Computing and Network (UCN) Project, Knowledge and Economy Frontier RD Program of the Ministry of Knowledge Economy (MKE) in Korea and a result of subproject UCN 09C1-C5-20S.

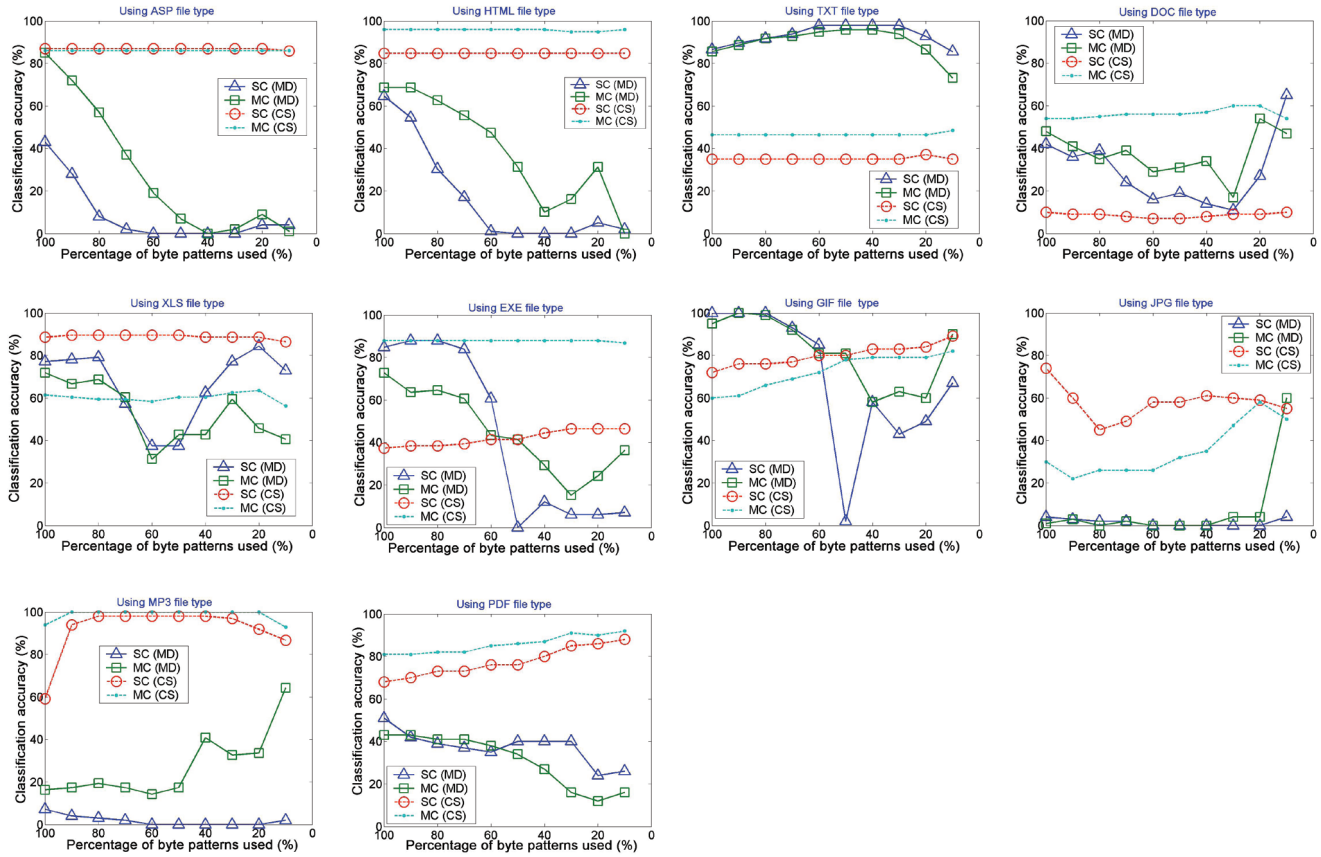
H Shin would like to gratefully acknowledge support from Post Brain Korea 21 and the research grant from Korean Government (MOEHRD).

References

1. Exclusion option to skip the files for scan in Norton antivirus, http://service1.symantec.com/SUPPORT/nav.nsf/0/c829006aa01d540b852565a6007770d8? [last cited on 2009 Sep 25].
2. Stegdetect. Available from: http://packages.debian.org/unstable/utils/stegdetect. [last cited on 2009 Sep 25].
3. Libmagic1 package. Available from: http://packages.debian.org/unstable/libs/libmagic1 [last cited on 2009 Sep 25].
4. W.J. Li, K. Wang, S.J. Stolfo, and B. Herzog. "Fileprints: Identifying file types by n-gram analysis," in workshop on Information Assurance and security (IAW'05), United States Military Academy, West Point, NY, pp. 64-71, 2005.
5. M. McDaniel, and M.H. Heydari. "Content based file type detection algorithms," in proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), pp. 332-42, 2003.
6. K. Wang, and S.J. Stolfo. "Anomalous payload-based network intrusion detection," in International Symposium on Recent Advances in Intrusion Detection (RAID'04), pp. 203-22, 2004.
7. N. Srinivasan, and V. Vaidehil. "Reduction of false alarm rate in detecting network anomaly using mahalanobis distance and similarity measure," in proceedings of ICSCN, pp. 366-71, 2007.
8. I. Ahmed, and K.S. Lhee. "Detection of malcodes by packet classification," in Workshop on Privacy and Security by means of Artificial Intelligence (ARES'08), Spain, pp. 1028-35, 2008.
9. File extensions. Available from: http://www.file-extension.com/ [last cited on 2009 Sep 25].

10. Magic numbers. Available from: http://qdn.qnx.com/support/docs/qnx4/utils/m/magic.html [last cited on 2009 Sep 25].
11. C. Nachenberg. "Polymorphic virus detection module," United States Patent No. 5826013, 1998.
12. P. Szor, and P. Ferrie. "Hunting for metamorphic," in proceedings of Virus Bulletin Conference, pp.123-44, 2001.
13. RIX, Writing IA32 Alphanumeric Shellcodes, Available from: http://www.phrack.org/issues.html?issue=57andid=15#article [last cited on 2009 Sep 25].
14. R.M. Harris. "Using artificial neural networks for forensic file type identification," in technical report at Purdue University, USA, 2007.
15. R. Eller. "Bypassing MSB Data Filters for Buffer Overflow Exploits on Intel platforms," Available from: http://community.core-di.com/~juliano/bypassmsb.txt, [last cited on 2003].
16. B. Li, Q. Wang, and J. Luo. "Forensic analysis of document fragment based on SVM," in proceedings of International Conference on Intelligent Information Hiding and Multimedia, Pasaena, CA, pp. 236-9, 2006.
17. K. Martin, and S. Nahid. "Oscar - file type identification of binary data in disk clusters and RAM pages," in IFIP security and privacy in dynamic environments, pp.413-24, 2006.
18. P. Szor, and P. Ferrie. "Hunting for metamorphic," in proceedings of Virus Bulletin Conference, pp. 123-44, 2001.
19. M.C. Amirani, M. Toorani, and A.A.B. Shirazi. "A new approach to content-based file type detection," in IEEE Symposium on Computers and Communications (ISCC'08), pp.1103-8, 2008.
20. S. J. Moody, and R.F. Erbacher. "SADI - Statistical Analysis for Data type Identification," in third international workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'08), pp. 41-54, 2008.
21. G.A. Hall, and W.P. Davis. "Sliding window measurement for file type identification," Available from: http://www.mantechcfia.com/SlidingWindowMeasurementforFileTypeIdentification.pdf [last cited on 2009 Sep 25].
22. C.J. Veenman. "Statistical disk cluster classification for file carving," in IEEE third international symposium on information assurance and security, pp. 393-8, 2007.
23. A.N. Kolmogorov. "Three approaches to the quantitative definition of information," Problems of Information Transmission, vol. 1, pp. 1-11, 1965.
24. W.C. Calhoun, and D. Coles. "Predicting the types of file fragments," Digital Investigation, Elsevier, vol. 5(1), pp.14-20, 2008.
25. Ahmed, K.S. Lhee, H. Shin, and M.P. Hong. "On improving the accuracy and performance of content-based file type identification," in proceedings of the 14th Australian conference on information security and privacy (ACISP'09), Australia, pp.44-59, 2009.
26. P.N. Tan, M. Steinbach, and V. Kumar. Introduction to data mining. Addison Wesley, 2005.
27. J. Mena. Investigative data mining for security and criminal detection. Butterworth-Heinemann, 2003.
28. K. Wang, J.J. Parekh, and S.J. Stolfo. "Anagram: a content anomaly detector resistant to mimicry attack," in International Symposium on Recent Advances in Intrusion Detection (RAID'06), pp. 226-48, 2006.
29. G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. "BotHunter: detecting malware infection through ids-driven dialog correlation," in 16th USENIX Security Symposium, pp. 167-82, 2007.
30. J.H. Ward. "Hierarchical grouping to optimize an objective function," Journal of the American Statistical Association, vol. 58(301), pp. 236-44, 1963.
31. A.C. Rencher. Methods of Multivariate Analysis. Wiley-Interscience, 2002.
32. T.P. Liang, H. Moskowitz, and Y. Yih. "Integrating neural networks and semi-markov processes for automated knowledge acquisition," An Application to Real-time Scheduling, Decision sciences, vol, 23(6), pp. 1297-314, 2007.
33. R.O. Duda, P.E. Hart, and D.G. Stork. Pattern classification. John Wiley and Sons, 2001.
34. SAS E-miner. Available from: http://www.sas.com/ [last cited on 2009 Sep 25].

Appendix

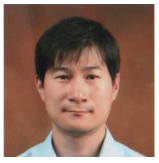


Comparison of single centroid (SC) and multicentroid (MC) models with the cosine similarity (CS) and Mahalanobis distance (MD).

AUTHORS

Irfan Ahmed received his B.E. degree from NED University of Engineering and Technology (NEDUET) in 2003 and M.S. degree in Computer Science from SZABIST, Pakistan in 2005. He is currently pursuing his Ph.D. degree at Graduate School of Information and Communication, Ajou University, South Korea.

E-mail: irfan@ajou.ac.kr



Kyung-suk Lhee received the Ph.D. degree in Computer and Information science from Syracuse University in 2005. He joined the Division of Information and Computer Engineering at Ajou University, Korea in 2005. His research interests are in the areas of computer and network security.

E-mail: klhee@ajou.ac.kr

Hyunjung (Helen) Shin received the Ph.D. degree in Data Mining from Seoul National University, and further majored in Machine Learning during her Post-Doc at Max Planck Institute in Germany. Since 2006, she joined Ajou University as a faculty member of the Department of Industrial and Information Systems Engineering. Her research activities range across areas as different as hospital fraud detection, direct marketing in CRM, oil/stock price prediction, bio-medical informatics, etc.

E-mail: shin@ajou.ac.kr

Manpyo Hong is a Professor in the Division of Information and Computer Engineering at Ajou University. His current research interests are Information Security. He received a BS, MS and PhD degree from the Department of Computer Science, Seoul National University, in 1981, 1983 and 1991 respectively.

E-mail: mphong@ajou.ac.kr

DOI: 10.4103/0256-4602.67149; Paper No TR 333_09; Copyright © 2010 by the IETE