Fast content-based file-type identification[☆]

Irfan Ahmed[a], Kyung-suk Lhee[b], Hyunjung Shin[c], ManPyo Hong[b]

[a]*Information Security Institute, Queensland University of Technology, Australia*
[b]*Digital Vaccine and Internet Immune System Lab, Graduate School of Information and Communication, Ajou University, South Korea*
[c]*Department of Industrial and Information Systems Engineering, Ajou University, South Korea*

Abstract

Content-based file-type identification schemes often use byte-frequency distribution as a feature and use statistical and data mining techniques to classify file types. Since those schemes use the entire file content to obtain byte-frequency distribution and use all possible byte patterns in file classification, they are inefficient and time-consuming. This paper proposes two techniques to reduce the classification time. The first method is a feature selection technique, which uses a subset of highly-occurring byte patterns in building the representative model of a file type and classifying files. To evaluate its effectiveness, we applied it to the six most popular classification algorithms (i.e. neural network, linear discriminant analysis, K-means, K-nearest neighbor, decision tree, and support vector machine). On average, the K-nearest neighbor method achieved the optimum accuracy of 90% using only 40% of byte patterns; this reduces 55% of computation time. The second method is the content sampling technique, which uses a small portion of a file to obtain its byte-frequency distribution. It is effective for large size files where a relatively small sample can generate the representative byte frequency distribution. For instance, it reduces the sampling size of MP3 files from 5MB to 400KB (without compromising the accuracy). This is a 15 fold size reduction.

Keywords: Byte frequency distribution, n-gram, Content classification, File type identification

[☆]A poster (preliminary work) has already been presented in the 25th ACM Symposium on Applied Computing (ACM SAC 2010) [1]

Email: irfan.ahmed@qut.edu.au, (klhee, shin, mphong)@ajou.ac.kr ()

1. Introduction

Identifying file types (ASP, JPG, EXE, etc.) is a non-trivial task that is required in many computer applications. For example, file carving [2, 3], which is a forensics technique, recovers files using file content. Many steganalysis programs, such as stegdetect [4] that detects steganographic contents in images, rely on file type identification as well.

File types are usually identified by the file extensions [5] or the magic numbers in the file header [6]. However, these methods can easily be deceived by changing the file extension or magic number. Therefore, especially in the presence of adversaries, a more reliable solution is needed. Analyzing file contents to find distinguishable patterns between different file types is a viable alternative but it is not wide spread, because it is inefficient and time-consuming. Existing approaches generate the byte-frequency distribution of a file and use it for classification with statistical or data mining techniques. The calculations required for the distribution may be time consuming because it scales with file size. Also, classifying file types with distributions may require a large memory space and computation time, since the computation time exponentially increases with the number of sequences of n-bytes (n-gram).

This paper proposes two approaches to reduce the classification time. Firstly, we propose a feature selection technique to reduce the memory space and the computation time of a classification algorithm. The idea is that a subset of high-frequency byte patterns may be sufficient to build the representative model of a file type. Using a subset of high-frequency byte patterns may even increase the classification accuracy, since low-frequency patterns are highly likely to be simple noise.

In the feature selection technique, we extract a certain percentage of high-frequency byte patterns from each type of files (we used the ten file types described in table 2). Since each file type shows a different set of high-frequency byte patterns, we merge the byte pattern sets of all file types into a unified feature set by using the *union* or *intersection* operation. We tested the proposed feature selection algorithm on the six most popular classification algorithms (classifiers): neural network, linear discriminant analysis, K-means, K-nearest neighbor (KNN), decision tree, and support vector machine. Our empirical results show that KNN achieves the highest accuracy of about 90% using only 40% of (1-gram) byte patterns.

Secondly, instead of using the entire file content, we propose a content sampling technique to reduce the time taken for obtaining the byte-frequency

distribution. To evaluate the effectiveness of this approach, we sample in two ways: 1) sampling initial contiguous bytes and 2) sampling a few small blocks in random locations in a file. Sampling initial contiguous bytes is also used by Li et al. [7] based on single-centroid, multi-centroid and exemplar files. However, their models are unable to identify similar file types such as text files (TXT, ASP and HTML) that only contain ASCII or printable characters. In this paper, we provide more general experiments to evaluate the sampling approach.

The rest of the paper is organized as follows. Section 2 presents the related work. The proposed techniques are described in Section 3. Section 4 describes the six classification algorithms used in our experiments. The empirical results are presented in Section 5, followed by the conclusions and future work in Section 6.

2. Related work

This section discusses various content-based file-type identification schemes based on the byte-frequency distribution.

McDaniel and Heydari [8] introduced three algorithms to analyze file content and identify file types. The byte-frequency analysis algorithm (BFA) computes the byte-frequency distribution of different files and generates a "fingerprint" of each file type by averaging the byte-frequency distribution of their respective files. To obtain another characterizing factor, they also calculate the correlation strength as by taking the difference between the same byte in different files. As the difference becomes smaller, the correlation strength approaches to 1, and vice versa. The byte-frequency cross-correlation algorithm finds the correlations between all byte pairs. It calculates the average frequencies of all byte pairs and the correlation strength in a similar manner to the BFA algorithm. The file header/trailer algorithm uses the byte-patterns of the file headers and trailers that appear in a fixed location at the beginning and end of a file, respectively. It maintains an array of size 256 for each location and the array entry corresponding to the byte is filled with the correlation strength of 1. It constructs the fingerprint by averaging the correlation strength of each file. In these algorithms, they compare the file with all the generated fingerprints in order to identify its file type.

Li et al. [7] used n-gram analysis to calculate the byte-frequency distribution of a file and build three different models of a file type, i.e. single

centroid (one model of each file type), multi-centroid (multiple models of each file type), and exemplar files (set of files of each file type) as centroid. They refer to them as a "fileprint". The single and multi-centroid models compute the mean and standard deviation of the byte-frequency distribution of the files of a given file type, which is then used for the Mahalanobis distance to find the file type with the closest model. In the exemplar file model, they compare the byte frequency distribution of exemplar files with that of the given file (there is no variance computed), and the Manhattan distance is used instead of the Mahalanobis distance. Their solution cannot identify files having similar byte-frequency distributions such as MS Office file formats (e.g. Word and Excel) but instead treats them as a single group or abstract file type.

Martin and Nahid [9, 10] proposed the "Oscar" method to identify a file fragment type. They use the single centroid model [7] of Li et al. and use a quadratic distance metric and 1-norm as a distance metric to compare the centroid with the byte-frequency distribution of a given file. Although their method identifies any file type, they have specifically optimized it for JPG files. They reported a 99.2% detection rate with no false positives.

Veenman [11] uses linear discriminant analysis to identify file types. Three features are obtained from the file content, i.e. the byte frequency distribution, the entropy derived from the byte-frequency distribution of files, and the algorithmic or Kolmogorov complexity that exploits the substring order [12]. Calhoun and Coles [13] extended Veenman's work and built classification models that are based on the ASCII frequency, entropy, and other statistics and apply linear discriminant analysis to identify file types. They also argued that the files of same type probably have longer common substrings than those of different types. Veenman reported a 45% overall accuracy.

Harris [14] uses neural networks to identify file types. He divides the files into blocks of 512 bytes, and uses only the first 10 blocks for file-type identification. Two features are obtained from each block, i.e. raw filtering and the character code frequency. Raw filtering takes each byte as an input to a single neuron of the neural network. On the other hand, the character code frequency counts how many times each character code occurs in the block and takes the frequency of the character as the input to the neurons. It is assumed that raw filtering is useful for files whose byte patterns would occur at regular intervals, while the character code frequency is useful for files which have irregular occurrences of byte patterns. He used only image files (JPG, PNG, TIFF, GIF, and BMP) as a sample set and reported a

detection rate from 1% (GIF) to 50% (TIFF) when using raw filtering and from 0% (GIF) to 60% (TIFF) when using the character code frequency.

Amirani et al. [15] use a hierarchical feature-extraction method to better exploit the byte-frequency distribution of files in file-type identification. They believe that the multiplicity of features degrades the speed and accuracy in file-type identification. Thus, they utilize principal component analysis and auto-associative neural networks to reduce the 256 features of byte patterns to a smaller number (where the detection error is negligible). After feature extraction, they use a three layer MLP (multi layer perceptron) for detecting the file types. They used DOC, PDF, EXE, JPG, HTML and GIF file types for the experiments (each having 30 test data points) and reported an accuracy of 98.33%.

In our paper [16], we proposed two approaches to improve the accuracy of file-type classification. The first approach compares the cosine similarity and Mahalanobis distance to show that the cosine similarity is a better comparison metric in terms of the detection speed and classification accuracy. It optimizes the processing time by using a subset of high-frequency byte patterns. This method is different from the proposed method in that it uses a different set of byte patterns for each file type, whereas our method combines the sets of byte patterns (using the union or intersection operation) to make a single set of features for identifying all file types. Another difference is that it only uses the cosine similarity and Mahalanobis distance for classification, whereas our method uses the six most popular classifiers and shows their accuracy in order to prove the effectiveness of the feature selection approach in general. The second approach groups the files irrespective of file types and then applies linear discriminant analysis to subsequently distinguish files in each group for better classification accuracy. We reported 70% and 77% overall classification accuracy using the first and second approaches, respectively.

3. Proposed approaches

This section describes the two proposed approaches for fast file-type identification. 1) The feature selection technique reduces the classification time. 2) The (file) content sampling technique reduces the time taken to obtain the byte-frequency distribution, which is used by the classifier.

% of high frequency byte patterns	Number of patterns after Union	% of features selected (Union)	Number of patterns after Intersection	% of features selected (Intersection)
10	102	39.84	-	-
20	155	60.54	-	-
30	202	78.90	2	0.78
40	236	92.18	6	2.34
50	247	96.48	15	5.85
60	253	98.82	33	12.89
70	256	100	51	19.92
80	-	-	75	29.29
90	-	-	122	47.65
100	-	-	256	100

Table 1: Percentages of high-frequency byte patterns (features) per file type, and the corresponding sets of (unified) features that are chosen by the union and intersection operation.

3.1. Feature selection technique

Assuming that a few of the most frequently occurring byte patterns may be sufficient to represent the file type, we propose to use a subset of high-frequency byte patterns as features. Since each file type has a different set of high-frequency byte patterns, we merged the sets of patterns into a unified set of features for all file types. This will be fed into the classifier.

For merging we used two strategies: union and intersection. The union combines the feature sets of all file types, and the intersection extracts the common set of features among the file types. The result of the union operation may include low-frequency byte patterns for certain file types, if those patterns occur frequently in other file types. In contrast, the result of the intersection operation guarantees that only the high-frequency byte patterns are included, but some of them will be omitted if they do not occur frequently in all file types.

3.2. Content sampling technique

Obtaining the byte-frequency distribution may be hugely time consuming if the whole file is used. Instead of using the entire file content, and assuming that partial file content may be enough to generate a representative byte-frequency distribution of the file type; we propose to sample the file content to reduce the time taken to obtain the byte-frequency distribution.

To evaluate the effectiveness of this approach, we sample in two ways: 1) sampling initial contiguous bytes 2) sampling a few small blocks in random locations in a file. The first method is the fastest way of sampling, but the

obtained data are location-dependent and hence may be biased. The second method gathers location-independent data and thus is free from such problems, but is slower than the first method (albeit much faster than using the whole file) because files are sequentially accessed medium. We can intuitively say that the second method (random sampling) can generate a better byte frequency distribution because the range of sampling covers whole the file. Thus, it can achieve better classification accuracy while keeping the sample size the same. We explain the accuracy of this technique later in section 5.3.

Random sampling is novel in file type identification, however, initial contiguous bytes sampling was also used by Harris [14] and Li et al. [7]. Harris randomly chose the sample size of 512 bytes. On the other hand, Li et al. took different sample sizes of upto 1000 bytes and showed that as the sample size increased the classification accuracy decreased. They achieved the optimum accuracy when using the initial 20 bytes of a file (i.e. file header). Thus, intuitively we can say that their scheme is header-dependent. However, our sampling technique is header-independent because we consider a relatively large block of a file (400KB was found to be the optimum) to generate the byte frequency distribution, where the few header bytes cannot change the distribution. In this paper, we argue about the optimum block size and whether the content sampling is effective for all given file types. To investigate, we increase the block size until we obtain a consistent accuracy for a given file type or until the whole file content is sampled.

4. Classification Algorithms

To prove the effectiveness of the feature selection technique, we tested it on the six most popular classifiers: neural network, linear discriminant analysis, k-means, k-nearest neighbor, decision tree, and support vector machine. This section explains how these algorithms are applied to file-type identification. The classification accuracy is computed using the following measure.

$$Accuracy(\%) = \frac{(Number\ of\ correctly\ predicted\ files) \times 100}{Total\ number\ of\ files} \tag{1}$$

To normalize the file sizes, we scaled their byte-frequency distribution from 0 to 1 by dividing each file's byte-pattern frequency by the file size (relative byte frequency).

4.1. Neural network (NN)

The neural network [17] is a non-linear classifier. We use a three-layered network. It has 256 input nodes and 6 hidden nodes. The 256 input nodes represent the byte patterns whose frequencies are passed as input values to the nodes. The number of hidden nodes is set to 6 as there is no further improvement seen in the classification accuracy. The activation function is a hyperbolic tangent and the learning rate is 0.1 as in Dua et al.[18].

4.2. Linear discriminant analysis (LDA)

Linear discriminant analysis (LDA) [19] finds linear combinations of byte patterns by deriving a discriminant function for each file type. The discriminant function is used to identify the type of test file. The output of a linear discriminant function, the discriminant score, is used to identify the file types.

4.3. K-means

K-means [17] computes one centroid for each file type by averaging the byte frequency distribution of the sample files of each file type. We also calculate the standard deviation as another characterizing factor of file types in order to compute Mahalanobis distance. At the detection time, it uses the Mahalanobis distance and computes the distance between the test file and the centroids of all file types. The file type having the least distance from its centroid to the test file is considered to be the file type of the test file.

4.4. Decision tree (DT)

The decision tree [17] maps the byte-frequency patterns into a tree structure reflecting the file types. In other words, the internal nodes correspond to the 256 byte patterns and the leaf nodes correspond to the file types. In the learning phase, the tree is built by selecting the byte patterns that best split the training files into their true file types. In the prediction phase, a test file traverses the tree from the root to the leaf nodes. The file type corresponding to the leaf node (given in the learning phase) is the predicted file type of the test file.

4.5. K-nearest neighbor (KNN)

K-nearest neighbor [17] is a lazy learner, which implies that it uses the sample files only when a test file is given. It calculates the distance of the test file from other sample files (the Manhattan distance is the metric). The majority file type among the k nearest files is the file type of the test file. The classification accuracy is calculated for all values of k i.e. from 1 to the number of sample files and the value of k corresponds to the highest classification accuracy.

4.6. Support vector machine (SVM)

The support vector machine [20, 17] is a linear machine working in a high-dimensional nonlinear feature space, and separating two classes by finding a hyperplane with a maximal margin between them. In case when the classes are not linearly separable in the original input space, it first transforms the original input space into a high dimensional feature space.

Given a training set with instances and class-label pairs (x_i, y_i) where i=1,2,...,m and $x_i \epsilon\ R^n, y_i \epsilon \{1, -1\}^m$, the function ϕ maps the training vector x_i into a high dimensional space to find a linear separating hyperplane with a maximal margin. The kernel function used for transformation can be defined as $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$.

After learning the models, the test instances are predicted. Let $z_1, z_2, ..., z_k$ be the k test instances and $f(z_1), f(z_2), ..., f(z_k)$ be their predicted decision values. If the true target values of test instances are known and represented as $y_1, y_2, ..., y_k$, the classification accuracy is measured as follows:

$$Accuracy = \mid i | y_i f(z_i) > 0 \mid /(k \times 100) \tag{2}$$

We aim to classify r multiple file types where $r > 2$. Since SVM is originally a model for binary classification, the "one-against-one approach" [21] is used for multiple-class classification. For r file types, it constructs $f_i(x)$ classifiers where $i = 1, 2, ..., r(r-1)/2$. Each classifier trains data from two different file types. After training, we need to aggregate several independently trained SVMs to identify a true file type of a test instance. For this we use majority voting. Let f_i be the decision function of the SVM_i and $C_j (j = 1, 2, ..., C)$ be the j^{th} class label, and N_{kj} be the number of SVMs whose decisions select the j^{th} class i.e. $N_{kj} = |\{(i | f_i(x_k) = C_j)\}|$. The final decision for test instance x_k is determined by majority voting as follows: $N_k = max(N_{kj})$.

S.No.	File type	Quantity	Average Size (Kilo Bytes)	Minimum Size (Bytes)	Maximum Size (Kilo Bytes)
1.	ASP	500	3.52	49	37
2.	DOC	500	306.44	219	7,255
3.	EXE	500	522.71	882	35,777
4.	GIF	500	3.24	64	762
5.	HTML	500	11.59	117	573
6.	JPG	500	1,208.27	21,815	7,267
7.	MP3	500	6,027.76	235	30,243
8.	PDF	500	1,501.12	219	32,592
9.	TXT	500	269.03	16	69,677
10.	XLS	500	215.98	80	9,892

Table 2: Details of dataset used for experiments

5. Empirical results

5.1. Dataset

We used 10 file types (HTML, PDF, JPG, EXE, GIF, TXT, DOC, MP3, ASP, and XLS) and there were 500 files of each type (refer to Table 2 for more details). 60% and 40% of the dataset was used as the training and test dataset for learning of the given algorithms and testing their classification accuracies, respectively. The file types in the dataset were chosen because they are popularly used and cover a broad range of file types including documents, executable and compressed files.

We collected the sample files from different sources to ensure that the sample files of a file type are not generated by one source. Thus, the executable files are mostly obtained from the bin and system32 folders from the Linux and Windows XP operating systems, respectively. Moreover, the other files are collected from the internet using a general search on Google. For example, .txt file was searched using option "filetype:txt". The image files such as GIF and JPG are also obtained from photo sharing websites such as picassa of google and flickr etc. The MP3 files are collected from different random sources mostly from the publicly available FTP servers for movies and personal computers. In short, the random collection of files can be considered an unbiased and representative sample of the given file types.

5.2. Feature selection

The given classifiers are applied to the dataset in order to compare their classification accuracies. The proposed feature selection technique is used to decrease the number of features for efficient detection without compromising the accuracy.

SVM Types	Kernel Types			
	Linear	**Polynomial**	**RBF**	**Sigmoid**
C-SVM	59.07	29.69	37.68	79.42
nu-SVM	89.80	29.68	84.86	79.41
One class SVM	0	0	13.94	0

Table 3: Comparison of SVM accuracy with respect to kernel types.

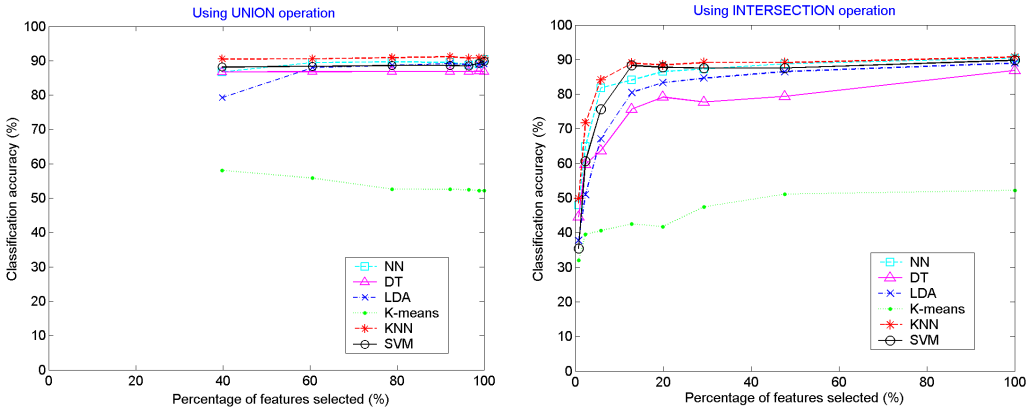

Figure 1: Average classification accuracies of the six classifiers. Features are selected using the union and intersection operations.

When using SVM, the choice of the SVM and kernel type is nontrivial with respect to the optimum classification results. Currently, we use four basic kernel functions (linear, polynomial, radial basis function, and sigmoid) and three SVM types (C-SVM, nu-SVM and one class SVM). Table 3 shows the SVM comparison with respect to the kernel type. It is shown that nu-SVM performs the best with the linear kernel type. Thus, for comparing SVM with other given algorithms, we use nu-SVM with linear kernel type.

We use the union and intersection to obtain a single unified set of byte patterns for all file types. It is noticed that the intersection operation obtains fewer features when the same percentage of high frequency byte patterns are used. For instance, for 30% of high frequency patterns (refer to Table 1), after the union the resultant set contained 79% of patterns, however, the intersection selected only two patterns which is 0.78% of all patterns.

We find the classification accuracy of all given file types for different percentages of highly-occurring byte patterns in order to find the optimum

File types	NN	DT	LDA	K-means	KNN	SVM
ASP	99	98	99	32.5	98.5	95.5
DOC	88	80.5	81.5	58	87	75.5
EXE	99	95.5	91	91.5	96	87
GIF	99.5	94.5	90.5	100	91	90.5
HTML	53	60	60	18.5	73.5	66
JPG	67	84	92	8	90	92.5
MP3	98	93.5	98	14	99.5	96
PDF	94	94	90.5	78	97	95
TXT	80	79	3	95.5	90	88.5
XLS	89.5	88.5	87	84.5	82.5	95

Table 4: The accuracies of classifiers for each file type, using 40% of the byte patterns obtained by the union operation. The circled cells show the highest accuracy for each file type.

accuracy for a subset of byte patterns and determine whether it remains reasonably stable when the highly-occurring byte patterns are increased in the subset. Figure 1 shows the average classification accuracies of the classifiers (using the features obtained from the union and intersection). It is noticed that the union shows more consistent accuracy than the intersection as the number of highly-occurring byte patterns increases. It is quite obvious, because unlike the intersection, the union contains all the highly-occurring byte patterns of all given file types and we assume (refer to section 3.1) that highly-occurring byte patterns are sufficient to represent a file type. Figure 1 shows that on average, KNN is the most accurate classifier among the tested algorithms. It produces about 90% accuracy using 40% of the features obtained using the union operation. If we use the intersection operation, we can further reduce the number of features without compromising the accuracy much. For instance, we can achieve an 88.45% accuracy using only 20% of the features.

Table 4 and 5 show that there is no single classifier which performs best for all given file types. However for most of the given file types the given classifiers (except K-means) achieve a similar classification accuracy. K-means is the simplest among all the given classifiers. It builds a single centroid for each file type by averaging the byte frequency distribution of its files. We investigated why the k-means classifier cannot achieve as higher accuracy as the other ones. It is found that the files of a file type can have different byte frequency distributions and averaging these results an inaccurate representative model of the file type. Thus, it results in a high misclassification rate.

At this point, it is important to understand why none of the given clas-

File types	NN	DT	LDA	K-means	KNN	SVM
ASP	93	95.5	99.5	26	98.5	91.5
DOC	87.5	84.5	74.5	40	88.5	77
EXE	96.5	83	91	73.5	97	87.5
GIF	95	80	88	79	82	93
HTML	53	68	66.5	6.5	74	63.5
JPG	89	87.5	89	5.5	87.5	90.5
MP3	98	93	98	5	99.5	95.5
PDF	92	88.5	74	67	94.5	95
TXT	79.5	86.5	76.5	98.5	82.5	87
XLS	89	10.5	89.5	73.5	87.5	95

Table 5: The accuracies of classifiers for each file type, using 40% of the byte patterns obtained by the intersection operation. The circled cells show the highest accuracy for each file type.

sification algorithms succeeded in accurately classifying the given file types. We obtain the confusion matrix of the given file types for 40% percentages of highly occurring byte patterns using the NN, DT and KNN classifiers (refer to Table 6). It is found that the similar types of files can be confused with each other. For instance; HTML files can be confused with ASP and TXT files and compound files such as XLS and DOC can be confused with each other.

In short, many classifiers show an accuracy of about 90% using 40% of features and this level of accuracy remains almost the same when the number of features is increased. Therefore we can conclude that the feature selection approach is effective. However, the accuracy of classifiers is dependent on the file types in question. If the given set contains confusing file types, it is highly likely that the accuracy will be low.

5.3. Content sampling

Since KNN performs best and achieves an optimum accuracy for 40% of features (selected using the union operation) (refer to Figure 1), we used KNN (with 40% of byte patterns) in our experiments for the content sampling technique.

Figure 2 and 3 show the classification accuracies for ten file types, which are grouped into three categories: binary group, text group, and binary-text group. These groups contain binary, ASCII or printable characters and compound files, respectively.

Figure 2 shows the result of initial contiguous byte sampling. It is noticed that the classification accuracy of file types shows extreme deviation i.e. either 0% or 100% when using the initial two bytes of a file. In general, the use of the first two bytes of a file is more likely to be signature matching,

File types	Classifier	ASP	DOC	EXE	GIF	HTM	JPG	MP3	PDF	TXT	XLS
ASP	**NN**	94	0.5	0	0	4	0	0	0	1.5	0
	KNN	98.5	0	0	0	1.5	0	0	0	0	0
	DT	98	1.5	0	0	0.5	0	0	0	0	0
	Average	96.83	0.66	0	0	2.0	0	0	0	0.5	0
DOC	**NN**	0	89	2.5	0.5	0.5	1	0.5	0	5.5	0.5
	KNN	0	87	2	0	0.5	1	0	0	4	5.5
	DT	0	80.5	0.5	2.5	2	0.5	1	0	3	10
	Average	0	85.5	1.66	1	1	0.83	0.5	0	4.16	5.33
EXE	**NN**	0	1.5	96	0	0	0.5	0	0	0.5	1.5
	KNN	0	3	96	0	0	0	0	1	0	0
	DT	0	1.5	95.5	0	0	0	1	0	0	2
	Average	0	2	95.83	0	0	0.16	0.33	0.33	0.16	1.16
GIF	**NN**	0	1	3.5	91	0	0	4	0	0	0.5
	KNN	0	6	0.5	91	0	0	1	0	0	1.5
	DT	0	1	0.5	94.5	0	0.5	2	0	0	1.5
	Average	0	2.66	1.5	92.16	0	0.16	2.33	0	0	1.16
HTML	**NN**	6.5	1	0	0	68	0	0	1	15	8.5
	KNN	5.5	0.5	0	0	73.5	0	0	1	19.5	0
	DT	15	0	0	0	60	0	0	0.5	14	10.5
	Average	9	0.5	0	0	67.16	0	0	0.83	16.16	6.33
JPG	**NN**	0	1	0	0.5	0	92.5	3.5	2	0	0
	KNN	0	3.5	3	0	0	90	1.5	0.5	0	1
	DT	0	1.5	0	7.5	0	84	5.5	0	0	1
	Average	0	2	1	2.66	0	88.83	3.5	0.83	0	0.67
MP3	**NN**	0	0	0	0	0	0	98	0	2	0
	KNN	0	0.5	0	0	0	0	99.5	0	0	0
	DT	0	0	0	4.5	1	0	93.5	0	1	0
	Average	0	0.16	0	1.5	0.33	0	97	0	1	0
PDF	**NN**	0	0	1	0.5	1	0.5	0.5	95.5	0.5	0.5
	KNN	0	0	0	0	0	1	0	97	1.5	0.5
	DT	0	0.5	0	2.5	1	1	0	94	0	1
	Average	0	0.16	0.33	1	0.67	0.83	0.16	95.5	0.67	0.67
TXT	**NN**	0.5	0	0	0	4.5	0	0	0.5	90	3
	KNN	1	0.5	0.5	1	3.5	0	0	1.5	90	2
	DT	1	2.5	0	0	16.5	0	0	1	79	0
	Average	0.83	1	0.16	0.33	8.16	0	0	1	86.33	1.67
XLS	**NN**	0.5	2.5	1	1	0	0	0	1.5	2.5	91
	KNN	0	13.5	0.5	0	0.5	0.5	0	2	0.5	82.5
	DT	0.5	8	0	0	0.5	0	0	0	2.5	88.5
	Average	0.33	8	0.5	0.33	0.33	0.16	0	1.16	1.83	87.33

Table 6: Confusion matrix using 40% of byte patterns obtained by the union operation.

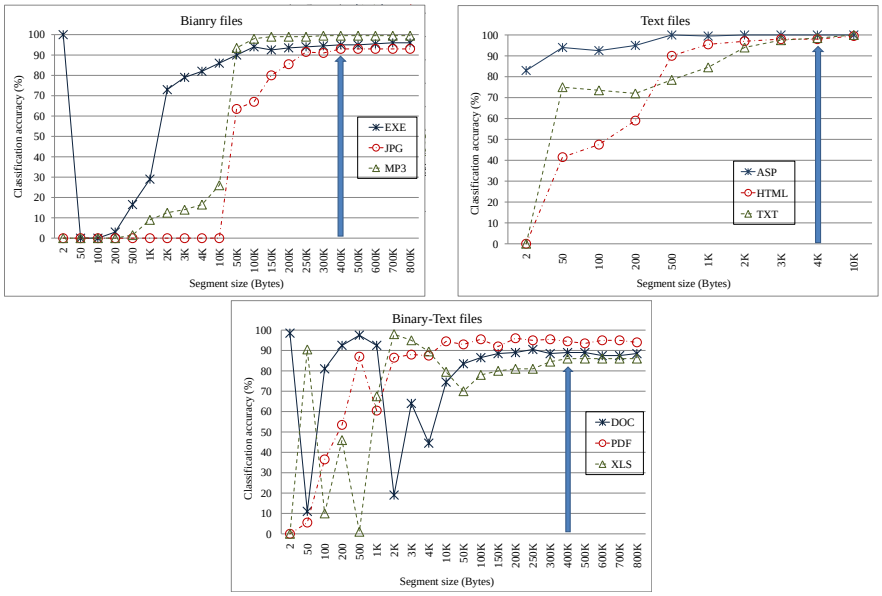

Figure 2: Classification accuracy using the initial contiguous bytes as the sampled content.The arrow shows the possible threshold value.

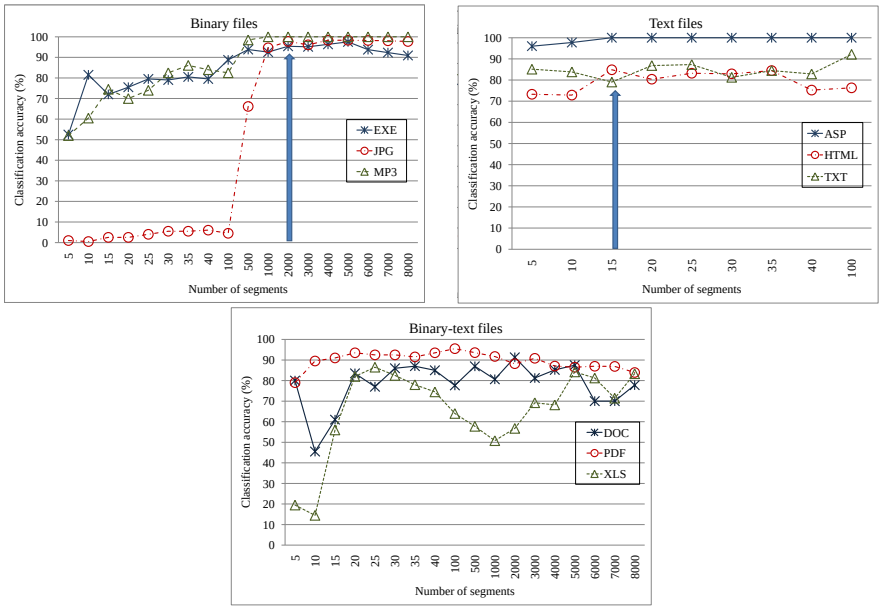

Figure 3: Classification accuracy using a small number of blocks in random locations as the sampled content (each block size is 100 bytes). The arrow shows the possible threshold value. (It is difficult to find an obvious threshold for binary-text files)

because the first two bytes of binary files such as EXE and JPG contain magic numbers. For instance, JPG and GIF files begin with "*FF D8*" and (*GIF89a* or *GIF87a*) respectively. Although the given text files do not have magic numbers, they often start with keywords. For instance, since HTML files are web pages created using scripting languages, they usually start with $< html >$ and $<!DOCTYPE$. In short, the given file types have certain patterns for their first two bytes. If the patterns indeed occur frequently and thus are included in the subset of 40% of byte patterns, it achieves 100% classification accuracy otherwise the given classifier fails to identify them. It is also noticed that the classification accuracy improves with the increase of initial contiguous bytes and becomes reasonably stable beyond a certain point. The maximum threshold value of the contiguous bytes found for the given file types is 400KB. This is significantly smaller than the average size of some of the given file types in the dataset. For instance, for the JPG, PDF and MP3 files, it is 3, 4 and 15 times smaller than their original size, which usually is 1200KB, 1500KB and 6000KB respectively.

Figure 3 shows the result of random sampling when small blocks (100 bytes in size) were sampled upto 8000 blocks.

Both schemes (initial contiguous bytes sampling and random sampling) achieve similar classification accuracy for binary and text files. However, unlike initial contiguous bytes sampling, random sampling fails to achieve a consistent accuracy in identifying compound files when the number of blocks increases. Thus, it is difficult to obtain a threshold value of a sample size for compound files. We conjecture that since a compound file can contain many embedded objects, random sampling generates different byte frequency distributions depending on the objects taken into account. If we compare the threshold values obtained by both sampling techniques, it is found that random sampling requires fewer bytes to achieve the optimal and stable accuracy in classifying binary and text files. It also corroborates our intuition that random sampling can generate a better byte frequency distribution of a file, because sampling involves the whole file.

In short, we conclude that content sampling is effective for large size files such as MP3 and JPG where relatively small samples can generate the representative byte frequency distribution.

5.4. Reduction in processing time

The total time taken to identify the file type includes 1) the time taken to obtain the byte-frequency distribution of the file, and 2) the time taken

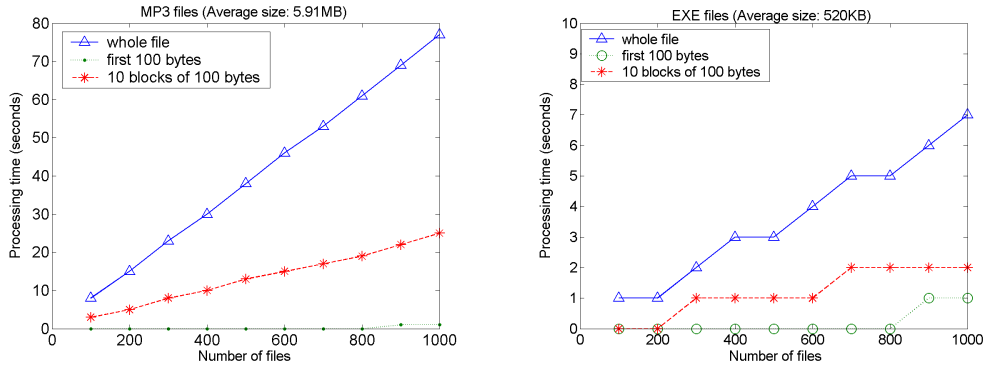

Figure 4: Reduction in processing time by the content sampling technique, which either used the initial contiguous 1000 bytes or ten 100-byte blocks in random locations.

by the classification algorithm to detect the file type. Since these two events are independent,in this section we will discuss them separately.

Figure 4 illustrates how much time could be saved in obtaining byte-frequency distributions by using the content sampling technique. Although this result was produced using the 1-gram, using a higher n-gram would yield similar results because the number of I/O operations would be the same regardless of the size of n.

Figure 5 illustrates how much time could be saved in the classification process (with KNN) by using the feature selection technique. Each algorithm has a different processing time depending on several factors such as whether it is a lazy or an eager learner, the number of attributes, or the technique used for comparison from the representative model. In this paper we show KNN as an example. KNN is a lazy learner in that it builds the representative model every time it needs to identify a file type. We used the Manhattan distance to find the distance between the test file and other sample files. Figure 5 shows that the KNN with the Manhattan distance achieves a 50% time reduction using 40% of the byte patterns.

We measured the time on a machine with 2.7Ghz Intel CPU and 2GB RAM running Windows XP.

6. Conclusions and future work

In this paper we presented two techniques for fast file-type identification. Firstly we proposed a feature selection technique that reduces the

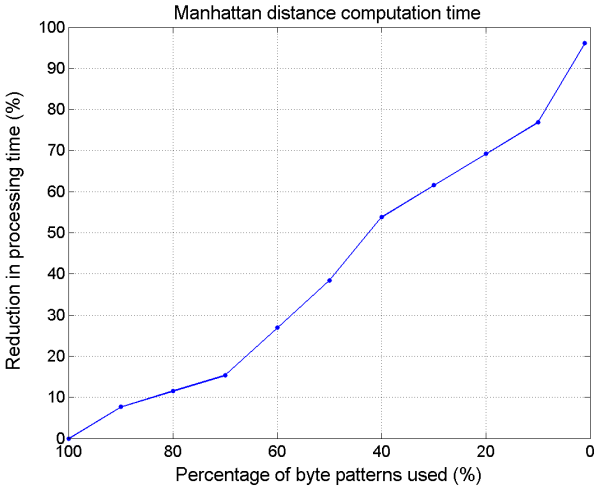

Figure 5: Reduction in processing Time by the feature selection technique in computing the Manhattan distance for the KNN classifier.

computation time of a classification algorithm, assuming that a subset of high-frequency byte patterns may be sufficient to build the representative model of a file type. Secondly, instead of using the entire file content, we proposed a content sampling technique that reduces the time taken to obtain the byte-frequency distribution of a file.

To evaluate the effectiveness of the proposed techniques, we conducted experiments using the six most popular classification algorithms to identify ten file types (HTML, PDF, JPG, EXE, GIF, TXT, DOC, MP3, ASP, and XLS).

Using the feature selection technique, many classifiers (KNN in particular) showed an accuracy of about 90% using only 40% of features; this is a 55% reduction in computation time. Therefore we conclude that the feature selection approach is highly effective. This technique saves substantial time. For example, KNN using the Manhattan distance can save about 50% of time by using 40% of the byte patterns.

The content sampling technique showed that a small block of a file is sufficient to generate the representative byte frequency distributions of the given file types. For instance, MP3 files have an average size of 5MB (in our dataset), and the empirical results showed that a 400KB sample of an MP3 file is enough to generate its byte frequency distribution (using initial contiguous bytes sampling) which is a 15-fold size reduction.

The proposed approaches showed promising results even with 1-gram features. We conjecture that we can achieve higher accuracy by increasing the size of the n-gram in order to obtain better features for the classifier. Moreover, substantial time can be saved by using a higher n-gram.

7. References

[1] I. Ahmed, K. suk Lhee, H. Shin, M. Hong, Fast file-type identification, in: ACM symposium on Applied computing, Sierre, Switzerland, 2010, pp. 1601–1602.

[2] S. L. Garfinkel, Carving contiguous and fragmented files with fast object validation, Digital Investigation 4 (1) (2007) 2–12.

[3] V. Roussev, S. L. Garfinkel, File fragment classification - the case for specialized approaches, in: Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, 2009.

[4] Stegdetect, `http://packages.debian.org/unstable/utils/stegdetect`.

[5] File extensions, `http://www.file-extension.com/`.

[6] Magic numbers, `http://qdn.qnx.com/support/docs/qnx4/utils/m/magic.html`.

[7] W. J. Li, K. Wang, S. Stolfo, 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, New York, USA, 2005, pp. 64–71.

[8] M. McDaniel, M. H. Heydari, Content based file type detection algorithms, in: proceedings of the 36th Annual Hawaii International Conference on System Sciences, Vol. 9, 2003, p. 332a.

[9] K. Martin, S. Nahid, Oscar: file type identification of binary data in disk clusters and ram pages, in: proceedings of IFIP International Information Security Conference: Security and Privacy in Dynamic Environments (SEC2006), 2006, pp. 413–424.

[10] S. N. Karresand Martin, File type identification of data fragments by their binary structure, in: proceedings of the 7th Annual IEEE Information Assurance Workshop, United States Military Academy, West Point, New York, USA, 2006, pp. 140–147.

[11] C. J. Veenman, Statistical disk cluster classification for file carving, in: IEEE third international symposium on information assurance and security, 2007, pp. 393–398.

[12] A. Kolmogorov, Three approaches to the quantitative definition of information, In Problems of Information Transmission 1 (1) (1965) 1–7.

[13] W. C. Calhoun, D. Coles, Predicting the types of file fragments, Digital Investigation 5 (1) (2008) 14–20.

[14] R. M. Harris, Using artificial neural networks for forensic file type identification, Technical report (May 2007).

[15] M. C. Amirani, M. Toorani, A. A. B. Shirazi, A new approach to content-based file type detection, in: IEEE Symposium on Computers and Communications, (ISCC '08), 2008, pp. 1103–1108.

[16] I. Ahmed, K. suk Lhee, H. Shin, M. Hong, On improving the accuracy and performance of content-based file type identification, in: 14th Australasian Conference on Information Security and Privacy, (ACISP'09), 2009, pp. 44–59.

[17] V. K. Pang-Ning Tan and, Michael Steinbach and, Introduction to Data Mining, Addison Wesley, US, 2005, Ch. Classification: Alternative Techniques.

[18] R. O. Duda, P. E. Hart, D. G. Stork, Pattern classification, 2nd Edition, John Wiley and Sons, New York, 2000, Ch. Multilayer neural network.

[19] A. C. Rencher, Methods of Multivariate Analysis, 2nd Edition, John Wiley and Sons, New York, 2002.

[20] S. Abe, Support Vector machines for pattern classification, 1st Edition, Springer, 2005.

[21] C. W. Hsu, C. J. Lin, A comparison of method of multi-class support vector machines, IEEE Transaction on Neural network 13 (2) (2002) 415–425.