Chapter 5

FAST CONTENT-BASED FILE TYPE IDENTIFICATION

Irfan Ahmed, Kyung-Suk Lhee, Hyun-Jung Shin and Man-Pyo Hong

Abstract Digital forensic examiners often need to identify the type of a file or file fragment based on the content of the file. Content-based file type identification schemes typically use a byte frequency distribution with statistical machine learning to classify file types. Most algorithms analyze the entire file content to obtain the byte frequency distribution, a technique that is inefficient and time consuming. This paper proposes two techniques for reducing the classification time. The first technique selects a subset of features based on the frequency of occurrence. The second speeds up classification by randomly sampling file blocks. Experimental results demonstrate that up to a fifteen-fold reduction in computational time can be achieved with limited impact on accuracy.

Keywords: File type identification, file content classification, byte frequency

1. Introduction

The identification of file types (e.g., ASP, JPG and EXE) is an important, but non-trivial, task that is performed to recover deleted file fragments during file carving [3, 12]. File carving searches a drive image to locate and recover deleted and fragmented files. Since the file extension and magic numbers can easily be changed, file type identification must only rely on the file contents. Existing file type identification approaches generate features from the byte frequency distribution of a file and use these features for classification [5, 9]. The problem is that this process requires considerable time and memory resources because it scales with file size and the number of n-gram sequences.

This paper presents two techniques that reduce the classification time. The first is a feature selection technique that selects a percentage of the most frequent byte patterns in each file type; the byte patterns for each

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics VII, IFIP AICT 361, pp. 65–75, 2011.

file type are then merged using a union or intersection operator. The second technique compares a sampling of the initial contiguous bytes [9] with samplings of several 100-byte blocks from the file under test.

Experimental tests of the techniques involve six classification algorithms: artificial neural network, linear discriminant analysis, k-means algorithm, k-nearest neighbor algorithm, decision tree algorithm and support vector machine. The results of comparing ten file types (ASP, DOC, EXE, GIF, HTML, JPG, MP3, PDF, TXT and XLS) show that the k-nearest neighbor algorithm achieves the highest accuracy of about 90% using only 40% of 1-gram byte patterns.

2. Related Work

Several algorithms have been developed to perform content-based file type identification using the byte frequency distribution. The byte frequency analysis algorithm [10] averages the byte frequency distribution to generate a fingerprint for each file type. Next, the differences between the same byte in different files are summed and the cross-correlation between all byte pairs is computed. The byte patterns of the file headers and trailers that appear in fixed locations at the beginning and end of a file are also compared. The file type is identified based on these three computed fingerprints.

Li, *et al.* [9] have used n-gram analysis to calculate the byte frequency distribution of a file and build three file type models (fileprints): (i) single centroid (one model of each file type); (ii) multi-centroid (multiple models of each file type); and (iii) exemplar files (set of files of each file type) as centroid. 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; the Mahalanobis distance is used to identify the file type with the closest model. In the exemplar file model, the byte frequency distributions of exemplar files are compared with that of the given file and the Manhattan distance is used to identify the closest file type. This technique cannot identify files that have similar byte frequency distributions such as Microsoft Office files (Word and Excel); instead, it treats them as a single group or abstract file type. Martin and Nahid [7, 8] have extended the single centroid model [9] using quadratic and 1-norm distance metrics to compare the centroid with the byte frequency distribution of a given file.

Veenman [14] has used three features: byte frequency distribution, entropy derived from the byte frequency and Kolmogorov complexity that exploits the substring order with linear discriminant analysis; this technique reportedly yields an overall accuracy of 45%. Calhoun and

Coles [2] have extended Veenman's work using additional features such as ASCII frequency, entropy and other statistics. Their extension is based on the assumption that files of the same type have longer common substrings than those of different types.

Harris [5] has used neural networks to identify file types. Files are divided into 512-byte blocks with only the first ten blocks being used for file type identification. Two features are obtained from each block: raw filtering and character code frequency. Raw filtering is useful for files whose byte patterns occur at regular intervals, while character code frequency is useful for files that have irregular occurrences of byte patterns. Tests using only image files (BMP, GIF, JPG, PNG and TIFF) report detection rates ranging from 1% (GIF) to 50% (TIFF) with raw filtering and rates ranging from 0% (GIF) to 60% (TIFF) with character code frequency.

Amirani, *et al.* [1] have employed a hierarchical feature extraction method to exploit the byte frequency distribution of files. They utilize principal component analysis and an auto-associative neural network to reduce the number of 256-byte pattern features. After feature extraction is performed, a multilayer perceptron with three layers is used for file type detection. Tests on DOC, EXE, GIF, HTML, JPG and PDF files report an accuracy of 98.33%.

3. Proposed Techniques

Two techniques are proposed for fast file type identification: feature selection and content sampling. Feature selection reduces the number of features used during classification and reduces the classification time. Content sampling uses small blocks of the file instead of the entire file to calculate the byte frequency; this reduces the feature calculation time.

Feature selection assumes that a few of the most frequently occurring byte patterns are sufficient to represent the file type. Since each file type has a different set of high-frequency byte patterns, classification merges the sets of most frequently occurring byte patterns. Merging uses the union and intersection operations. Union combines the feature sets of all the file types while 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. In contrast, the intersection operation guarantees that only the highest-frequency byte patterns are included.

Obtaining the byte frequency distribution of an entire file can be extremely time consuming. However, partial file contents may be sufficient to generate a representative byte frequency distribution of the file type.

The file content is sampled to reduce the time taken to obtain the byte frequency distribution.

The sampling effectiveness is evaluated in two ways: sampling initial contiguous bytes and sampling several 100-byte blocks at random locations in a file. The first method is faster, but the data obtained is location dependent and may be biased. The second method gathers location-independent data, but is slower because the files are accessed sequentially. Intuitively, the second method (random sampling) should generate a better byte frequency distribution because the range of sampling covers the entire file. Thus, it exhibits higher classification accuracy for a given sample size.

Random sampling is novel in the context of file type identification. However, initial contiguous byte sampling has also been used by Harris [5] and by Li, *et al.* [9]. Harris used a sample size of 512 bytes. Li, *et al.* employed several sample sizes up to a maximum of 1,000 bytes, and showed that the classification accuracy decreases with an increase in sample size. Optimum accuracy was obtained when using the initial twenty bytes of a file.

4. Classification Algorithms

Experimental tests of the two proposed techniques involve six classification algorithms: artificial neural network, linear discriminant analysis, k-means algorithm, k-nearest neighbor algorithm, decision tree algorithm and support vector machine.

Artificial neural networks [13] are nonlinear classifiers inspired by the manner in which biological nervous systems process information. The artificial neural network used in the experiments incorporated three layers with 256 input nodes and six hidden nodes. The 256 input nodes represented the byte frequency patterns. The number of hidden nodes was set to six because no improvement in the classification accuracy was obtained for larger numbers of nodes. A hyperbolic tangent activation function was used; the learning rate was set to 0.1 as in [4].

Linear discriminant analysis [11] finds linear combinations of byte patterns by deriving a discriminant function for each file type. The discriminant score produced as the output of the linear discriminant function was used to identify the file type.

The k-means algorithm [13] computes a centroid for each file type by averaging the byte frequency distribution of the sample files corresponding to each file type. In our experiments, the Mahalanobis distance between the test file and the centroids of all the file types was computed

by the k-means algorithms. The file type corresponding to the closest centroid was considered to be the file type of the test file.

The k-nearest neighbor algorithm [13] employs a lazy learning strategy that stores and compares every sample file against a test file. The Manhattan distance of the test file from all other sample files was calculated, and the majority file type among the k nearest files was considered to be the file type of the test file. The classification accuracy was calculated for values of k from one to the number of sample files, and the value chosen for k corresponded to the highest classification accuracy.

A decision tree algorithm [13] maps the byte frequency patterns into a tree structure that reflects the file types. Each node in the tree corresponds to a byte pattern that best splits the training files into their 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 of the tree was designated as the file type of the test file.

A support vector machine (SVM) [13] is a linear machine operating in a high-dimensional nonlinear feature space that separates two classes by constructing a hyperplane with a maximal margin between the classes. In cases when the classes are not linearly separable in the original input space, the original input space is transformed into a high-dimensional feature space.

Given a training set with instances and class-label pairs (x_i, y_i) where $i = 1, 2, \ldots, m$ and $x_i \epsilon R^n$, $y_i \epsilon \{1, -1\}^m$, the function ϕ maps the training vector x_i to a higher-dimensional space using a kernel function to find a linear separating hyperplane with a maximal margin. There are four basic kernel functions (linear, polynomial, radial basis function and sigmoid) and three SVM types (C-SVM, nu-SVM and one class SVM). Our preliminary tests determined that the best file classification performance was obtained using a nu-SVM with a linear kernel.

Since the SVM is a binary classifier, the one-versus-one approach [6] is used for multiclass classification. Thus, $r(r-1)/2$ binary classifiers must be constructed for r file types. Each binary classifier was trained using data corresponding to two file types. The final classification was determined based on the majority vote by the binary classifiers.

5. Experimental Results

The experimental tests used a data set comprising 500 files of each of ten file types (ASP, DOC, EXE, GIF, HTML, JPG, MP3, PDF, TXT and XLS) (Table 1). Classifier training used 60% of the data set while testing used the remaining 40% of the data set. The files came from different sources to eliminate potential bias. The executable files were

Table 1. Data set used in the experiments.

File Type	Number of Files	Average Size (KB)	Minimum Size (B)	Maximum Size (KB)
ASP	500	3.52	49	37
DOC	500	306.44	219	7,255
EXE	500	522.71	882	35,777
GIF	500	3.24	64	762
HTML	500	11.59	117	573
JPG	500	1,208.27	21,815	7,267
MP3	500	6,027.76	235	30,243
PDF	500	1,501.12	219	32,592
TXT	500	269.03	16	69,677
XLS	500	215.98	80	9,892

obtained from the `bin` and `system32` folders of Linux and Windows XP machines. The other files were collected from the Internet using a general search based on each file type. The random collection of files can be considered to represent an unbiased and representative sample of the ten file types.

5.1 Feature Selection

The feature selection tests sought to identify the merging operator, the percentage of features and the classifier with the best performance. The tests were designed to compare the six classifiers with different percentages of frequently occurring byte patterns and the union and intersection operators.

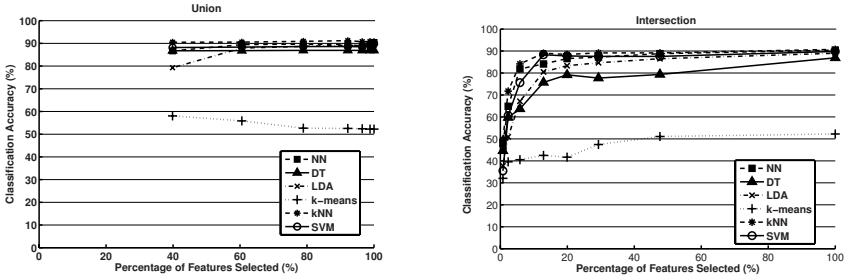

Figure 1. Average classification accuracy.

Figure 1 shows the average classification accuracy of each of the six classifiers for various percentages of the most frequently occurring byte patterns and the union and intersection operators. The union operator was more consistent with respect to accuracy than the intersection

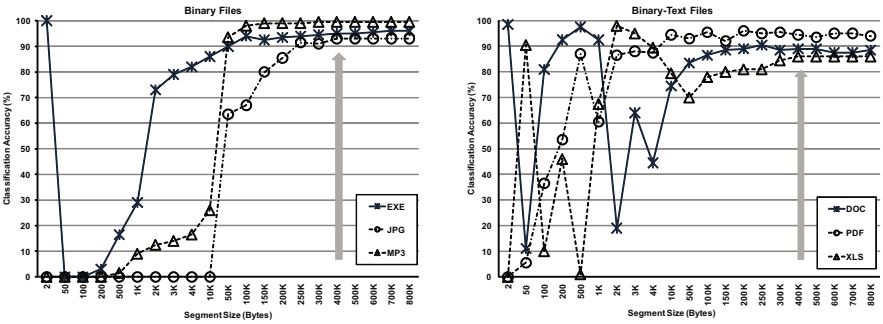

Figure 2. Classification accuracy using initial contiguous bytes as sampled content.

operator as the number of frequently occurring byte patterns increased. This occurs because union, unlike intersection, retains all the most frequently occurring byte patterns of the various file types. Additionally, as shown in Figure 1, the k-nearest neighbor (kNN) algorithm, on the average, yields the most accurate classifier of the six tested algorithms. In particular, it exhibits 90% accuracy using 40% of the features and the union operation. Using the kNN algorithm with the intersection operation further reduces the number of features without compromising the accuracy (88.45% accuracy using 20% of the features).

The results also show that no single classifier consistently exhibits the best performance. Many classifiers provide an accuracy of about 90% using 40% of the features, and this level of accuracy remains almost the same as the number of features is increased. This underscores the fact that the computational effort involved in classification can be reduced by using the most frequently occurring byte patterns for classification.

5.2 Content Sampling

This section focuses only on the results obtained with the kNN algorithm because it exhibited the best performance in our experiments involving feature selection.

Figures 2 and 3 show the classification accuracy for the ten file types that are divided into three groups: binary, text and binary text containing binary, ASCII or printable characters, and compound files, respectively. The arrows in the figures show the possible threshold values.

Figure 2 shows the results obtained for initial contiguous byte sampling. Note that the classification accuracy of file types shows an extreme deviation (either 0% or 100%) when the initial two bytes of a file are used. In general, the first two bytes of a file are more likely to match a signature because, in the case of binary files such as EXE and JPG,

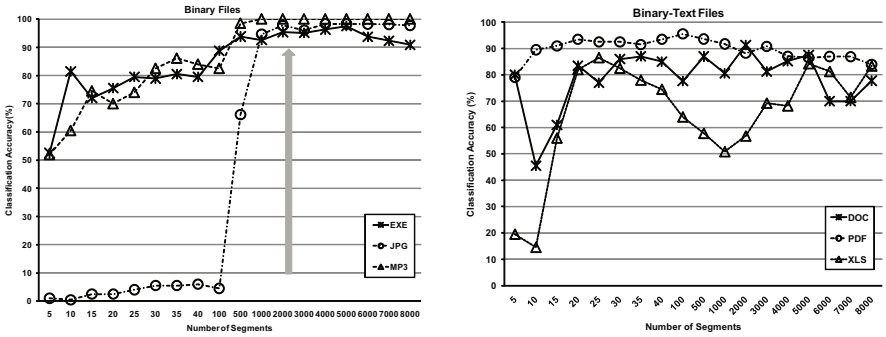

Figure 3. Classification accuracy using randomly-sampled 100-byte blocks.

these bytes contain magic numbers. For instance, JPG files begin with `FF D8` (GIF files begin with `GIF89a` or `GIF87a`). Although text files do not have magic numbers, they often start with keywords. For example, HTML files usually start with `<html>` and `<!DOCTYPE`.

In short, the first two bytes of file types have certain patterns. If the patterns occur frequently and are included in the subset of 40% of byte patterns, a classifier either identifies them with 100% accuracy or fails to identify them. Note also that the accuracy improves with an increase in the 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 400 KB. This is significantly smaller than the average size of the files in the data set. For example, the maximum threshold values for JPG, PDF and MP3 files are, respectively, three, four and fifteen times smaller than their original sizes.

Figure 3 shows the results obtained for random sampling of up to 8,000 100-byte blocks. Initial contiguous byte sampling and random sampling have similar classification accuracy for binary and text files. However, unlike initial contiguous byte 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 for the sample size for compound files. We conjecture that, because a compound file has many embedded objects, random sampling generates different byte frequency distributions depending on the objects that are taken into account. The comparison of the threshold values obtained by the two sampling techniques shows that random sampling requires fewer bytes to achieve the optimal and stable accuracy in classifying binary and text files. This also verifies that random sampling is effective for large files such as JPG and MP3 for which relatively small samples can generate the representative byte frequency distribution.

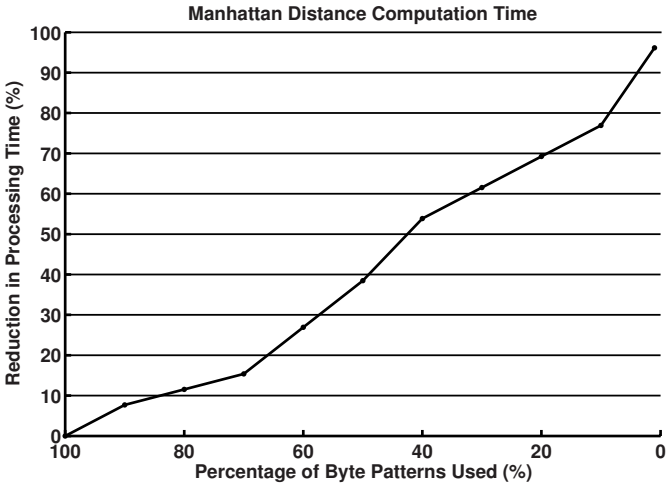

Figure 4. Time reduction using feature selection with a kNN classifier.

5.3 Time Reduction

The total time taken to identify a file type includes the time taken to obtain the byte frequency distribution of the file and the time taken by the classification algorithm to classify the file. The experimental tests undertaken to measure the time savings used a Windows XP machine with 2.7 GHz Intel CPU and 2 GB RAM.

Figure 4 illustrates the time savings that can be achieved in the classification process (with the kNN algorithm) by using the feature selection technique. Each algorithm has a different processing time depending on whether it uses lazy or eager learning, the number of attributes and the technique used for comparison with the representative model. Since kNN is a lazy learning algorithm and classification requires computations involving the test sample and all learned samples, the algorithm has high computational complexity with regard to classification. This makes the kNN algorithm a representative upper bound for the classification computational time. Figure 4 shows that the kNN algorithm with the Manhattan distance achieves a 50% time reduction using 40% of the byte patterns.

Figure 5 shows the computational time savings obtained when the byte frequency distribution is calculated using content sampling. Although the results were produced using 1-grams, a higher n-gram would yield similar results because the number of input/output operations is the same regardless of the size of n.

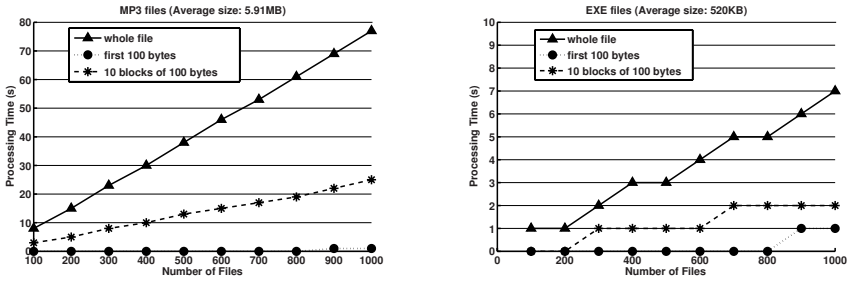

Figure 5. Time reduction using initial contiguous and 100-byte block sampling.

6. Conclusions

The two techniques described in this paper are designed to speed up file type identification. The first technique performs feature selection and only uses the most frequently occurring byte patterns for classification. The second technique uses random samples of the file being tested instead of the entire file to calculate the byte pattern distribution. Experimental tests involving six classification algorithms demonstrate that the kNN algorithm has the best file identification performance. In the best case, the proposed feature selection and random sampling techniques can produce a fifteen-fold reduction in computational time.

The proposed techniques yield promising results with 1-gram features. Higher accuracy can be achieved by increasing the n-gram size to obtain better features for classification. Using a higher n-gram can also result in significant time savings.

Acknowledgements

This research was performed under the Ubiquitous Computing and Network Project (UCN 10C2-C3-10M), which was supported by the Knowledge and Economy Frontier R&D Program of the South Korean Ministry of Knowledge Economy. This research was also partially supported by Grant No. 2010-0028631 from the National Research Foundation of South Korea.

References

[1] M. Amirani, M. Toorani and A. Shirazi, A new approach to content-based file type detection, *Proceedings of the Thirteenth IEEE Symposium on Computers and Communications*, pp. 1103–1108, 2008.

[2] W. Calhoun and D. Coles, Predicting the types of file fragments, *Digital Investigation*, vol. 5(S1), pp. 14–20, 2008.

[3] S. Garfinkel, Carving contiguous and fragmented files with fast object validation, *Digital Investigation*, vol. 4(S1), pp. 2–12, 2007.

[4] R. Duda, P. Hart and D. Stork, *Pattern Classification*, John Wiley, New York, 2001.

[5] R. Harris, Using Artificial Neural Networks for Forensic File Type Identification, CERIAS Technical Report 2007-19, Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, Indiana, 2007.

[6] C. Hsu and C. Lin, A comparison of methods for multiclass support vector machines, *IEEE Transactions on Neural Networks*, vol. 13(2), pp. 415–425, 2002.

[7] M. Karresand and N. Shahmehri, File type identification of data fragments by their binary structure, *Proceedings of the Seventh Annual IEEE Information Assurance Workshop*, pp. 140–147, 2006.

[8] M. Karresand and N. Shahmehri, Oscar – File type identification of binary data in disk clusters and RAM pages, *Proceedings of the IFIP International Conference on Information Security*, pp. 413–424, 2006.

[9] W. Li, K. Wang, S. Stolfo and B. Herzog, Fileprints: Identifying file types by n-gram analysis, *Proceedings of the Sixth Annual IEEE Information Assurance Workshop*, pp. 64–71, 2005.

[10] M. McDaniel and M. Heydari, Content based file type detection algorithms, *Proceedings of the Thirty-Sixth Annual Hawaii International Conference on System Sciences*, 2003.

[11] A. Rencher, *Methods of Multivariate Analysis*, John Wiley, New York, 2002.

[12] V. Roussev and S. Garfinkel, File fragment classification – The case for specialized approaches, *Proceedings of the Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 3–14, 2009.

[13] P. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining*, Addison-Wesley, Reading, Massachusetts, 2005.

[14] C. Veenman, Statistical disk cluster classification for file carving, *Proceedings of the Third International Symposium on Information Assurance and Security*, pp. 393–398, 2007.