Proceedings of the South African Information Security Multi-Conference (SAISMC 2010)

A Novel Support Vector Machine Approach to High Entropy Data Fragment Classification

Q. Li[1], A. Ong[2], P. Suganthan[2] and V. Thing[1]

[1]Cryptography & Security Dept., Institute for Infocomm Research, Singapore
e-mail: {qli,vriz}@i2r.a-star.edu.sg
[2]School of Electrical & Electronic Engineering, Nanyang Technological University, Singapore
e-mail: {ongy0035,epnsugan}@ntu.edu.sg

Abstract

A major challenge in digital forensics is the efficient and accurate file type classification of a fragment of evidence data, in the absence of header and file system information. A typical approach to this problem is to classify the fragment based on simple statistics, such as the entropy and the statistical distance of byte histograms. This approach is ineffective when dealing with high entropy data, such as multimedia and compressed files, all of which often appear to be random. We propose a method incorporating a support vector machine (SVM). In particular, we extract feature vectors from the byte frequencies of a given fragment, and use an SVM to predict the type of the fragment under supervised learning. Our method is efficient and achieves high accuracy for high entropy data fragments.

Keywords

Data classification, support vector machine, digital forensics

1. Introduction

An important task in digital forensics is the file type classification of a given evidence data fragment. This is an important step in many forensics applications. For example, when we try to recover important data from a hard disk with a corrupted partition table and/or file allocation table, it is desirable to identify the type of the data fragments found on the hard disk before trying to recover the data. In these applications, the accuracy of the classification is important since it allows us to reduce the scale of the problem (i.e., the number of data fragments in question) drastically and hence, making the analysis that follows much more efficient.

However, data fragment classification is challenging since there is no file header or file system information that could be referred to, which is typically the case when we are dealing with corrupted disks. Although it is often easy to distinguish between high entropy file types (e.g., compressed data) and low entropy types (e.g., text files and HTML files), the challenge arises when it is required to differentiate among high entropy file types.

A typical approach to data fragment classification is to examine its byte frequency histogram, i.e., the frequencies at which each of the byte values occurs in the data fragment, and sometimes other statistics. A statistical distance between the histogram and known distributions of different types of files can be computed and used to differentiate different data types. This approach is used, for example, in the Oscar method (Karrensand and Shahmehri, 2006), in combination with other methods. Nevertheless, the accuracy often suffers for very high entropy file types, where the histograms of different file types may look alike. In fact, it is acknowledged that, even when combined with various other approaches, it is still very difficult to differentiate among high entropy files such as JPEG images, executable EXE files and ZIP archives (e.g., Karrensand and Shahmehri, 2006, Veenman, 2007, and Moody and Erbacher, 2008).

Another method to determine the type of a given data fragment is to look for the existence (or the lack of) certain keywords that would only appear (or would never appear) in certain types of files. This method is often used in conjunction with the above statistical approach (e.g., Karrensand and Shahmehri, 2006). Although it works for high entropy file types where a certain byte pattern appears significantly more (or significantly less) often than others, it does not work well when there is no obvious pattern in the data.

Machine learning methods have recently been explored to tackle the data fragment classification problem. For example, Calhoun and Coles (2008) proposed a method based on Fisher's linear discriminant on a combination of a number of different statistics, which achieves reasonable accuracies at the cost of classification complexity.

In this paper we propose a novel method of data fragment classification through incorporation with Support Vector Machines (SVMs). SVMs are a very useful supervised learning tool that has been studied intensively recently. One of the advantages of SVM is that it allows us to exploit patterns in high dimensional spaces, which is very difficult or inefficient with conventional statistical methods.

In particular, we treat the histogram derived from a data fragment as its feature vector, and use a trained SVM to classify a given feature vector. Since the histogram is very easy to compute and the state-of-the-art SVM implementations are fast and highly accurate, our method is simple yet efficient and powerful.

We evaluate our technique by extracting a large number of feature vectors from the data files we collected, choosing optimal parameters for a given SVM implementation, and testing the performance against another set of testing files. We show that our method is both efficient and highly accurate. We note that our high accuracy is achieved by using only the histogram alone, without the combination of other methods. This demonstrates how well the SVM based method performs. It also allows further improvements on the classification accuracy by combining other techniques, such as the keyword based methods.

Our paper is organized as follows. In Section 2, we give a brief overview of related work. A short introduction to SVM is given in Section 3. We give a detailed description of our proposed method in Section 4. Evaluation of the performance of the proposed method is presented in Section 5. We conclude in Section 6.

2. Related Work

McDaniel and Heydari (2003) proposed three algorithms, Byte Frequency Analysis (BFA), Byte Frequency Cross-correlation (BFC) and File Header/Trailer (FHT), to generate the characteristic fingerprints to identify different computer file types. The BFA algorithm computes the frequency distribution of each file type by counting the number of occurrences of each byte value. The BFC algorithm considers the relationship between the byte value frequencies to strengthen the file type identification. The correlation strength between a byte pair is determined by the average difference in their frequencies. The FHT algorithm analyzes the file to look for byte pattern at fixed locations from the beginning and ending. The authors generated 30 file type fingerprints. They conducted experiments on 120 test files (i.e. 4 files for each file type) and considered files in whole only. The classification accuracy was 27.50%, 45.83% and 95.83% for the BFA, BFC and FHT algorithms, respectively.

Li et al. (2005) proposed enhancements to the BFA algorithm. They performed truncation to model a fixed portion of a file only. Multiple centroids (i.e. average vector of a set of vectors belonging to a cluster as defined by Damashek, 1995) for each file type were computed by clustering the training files using K-means algorithm. The authors also proposed using exemplar files as centroids by extending the multiple centroids method to compare the test data with all the training data (without applying K-means). They conducted experiments to classify 8 file types. However, exe and dll files were classified under the exe file category, while doc, ppt and xls files were considered as doc files. The classification accuracy approached 100% when only the first 20 bytes of the files were used to create the fingerprints. As the modeled portion increased in size, the accuracy dropped to 76.8% when classifying gif files in the multi-centroid technique, and 77.1% when classifying jpg files in the exemplar files technique. In these techniques, the modeled portions are always the beginning portion of each file (i.e. first 20, 200, 500 and 1000 bytes). Therefore, the header information (partial or full) is always included in the computation of the centroids, without which, the accuracy is expected to drop significantly.

Karresand and Shahmehri (2006) proposed a file type identification method, and named it Oscar. They built centroids of the mean and standard deviation of the byte frequency distribution of different file types. A weighted quadratic distance metric was used to measure the distance between the centroids and the test data fragments, so as to identify JPEG fragments. In addition, the detection capability of Oscar was enhanced by taking into consideration that byte 0xFF was only allowed in combination with a few other bytes (i.e. 0x00, 0xD0..D7) within the data part of JPEG files. Using a test data set of 17608 4KB blocks, the classification accuracy was 97.9%. The authors extended the Oscar method (Karresand and Shahmehri,

2006) by incorporating the byte ordering information through calculating the rate of change of the data bytes. Using a test data of 72.2MB, the classification accuracy for JPEG fragments was 99% with no false positive. However, for Windows executables, the false positive rate increased tremendously to exceed the detection rate (after the latter reached 12%). The detection rate for zip files was between 46% and 84%, with false positive rates in the range of 11% to 37%.

Hall and Davis (2006) proposed measuring the entropy and the data compressibility of the file using sliding windows of fixed sizes. The sliding window compressibility utilized the LZW algorithm. Instead of performing the actual compression, only the number of changes made to the compression dictionary was used so as not to compromise efficiency. Based on the computations, the average entropy and compressibility profiles were plotted for each file type. The authors then used the point-by-point delta and the Pearson's Rank Order Correlation computation to determine the file type of the test data. If the graph of the test data "fitted" the training data profiles, the file type would be considered as identified. The classification accuracy for zip files was 12%. The accuracy for the other file types was not presented. Instead, the authors re-evaluated the results to determine how often the correct file type was associated with a file within the top ten matching categories.

Erbacher and Mulholland (2007) studied the characteristics of 7 different file types by performing analysis on 13 statistics. Instead of identifying the file type of a file, their method was used to identify and locate presence of data of a particular type within a file or on a disk. The statistical measurements were plotted, and based on their observations they concluded that 5 of the statistics (i.e. average, kurtosis, distribution of averages, standard deviation and distribution of standard deviations) were sufficient to identify unique characteristics of each file type.

Veenman (2007) proposed combining the byte frequency histogram, the entropy information and the Kolmogorow complexity measurement to classify the file type of a cluster. The author used a training set of approximately 35,000 clusters and a test set of approximately 70,000 clusters. The clusters consisted of thirteen different file types. The classification accuracies were promising for the html, jpeg and exe types at 99%, 98% and 78%, respectively. However, the accuracies for the other file types were between 18% (for zip) and 59% (pdf).

Moody and Erbacher (2008) extended the work by Erbacher and Mulholland (2007) by considering the actual implementation of the technique and measured the accuracy of the classification method. The applied statistical measurements were based on the 5 most useful statistics derived by Erbacher and Mulholland (2007). The authors carried out the classification accuracy evaluation on 200 files of 8 different file types. The results indicated that the method alone could not differentiate between csv, html and text files reliably. They were all placed under the textual type class. The same applied to the dll and exe files. They were classified under the "dll and exe" category and achieved a 76% accuracy. The bmp, jpg and xls achieved accuracies of 64%, 68% and 4%, respectively. Additional analysis to identify unique characters within file types (such as a high number of matches for null data within xls files) was also performed. The accuracy for xls files was increased to 76%. The previously identified textual files were re-classified and the accuracies for csv, html and text files were 96%, 84% and 80%, respectively.

Calhoun and Coles (2008) proposed applying the Fisher linear discriminant to a set of different statistics (e.g. sum of the 4 highest byte frequencies, correlation of the frequencies of byte values), in addition to those proposed by Veenman (2007), as well as combinations of the statistics. They further enhanced the type classification capabilities of the technique by considering the common data subsequences within files of the same type. The evaluations were based on type pair comparisons between jpg, pdf, gif and bmp fragments (e.g. jpg vs. pdf, jpg vs. gif, pdf vs. bmp). The test set composed of 50 fragments of each file type. The fragment sizes were 896 (i.e. after removing the first 128 bytes from each of the 1024-byte fragments in the first experiment) and 512 bytes (after removing the first 512 bytes in another experiment). In the experiment with fragment size of 896 bytes, the combination of the Ascii-Entropy-Low-High-ModesFreq-SdFreq statistics achieved the highest average accuracy at 88.3%. Using the Ascii-Entropy-Low-High-ModesFreq-SdFreq statistics, the highest accuracy was achieved during the jpg vs. pdf classification at 98%, while the lowest was at 82% for the pdf vs. bmp classification. In the experiment with fragment size of 512 bytes, the longest common subsequence technique achieved the highest average accuracy at 86%. The highest accuracy was achieved during the pdf vs. bmp classification at 92%, while the lowest was for the jpg vs. bmp classification at 75%. A shortcoming of the longest common subsequence technique is the requirement to compare a test fragment to each of the sample files, thus increasing the classification complexity. In addition, the experiments considered comparisons between two file types at a time only. The test data set was also very limited, considering 50 fragments per file type.

3. Background on SVM

Support Vector Machines (SVMs) are machine learning algorithms that are very useful in solving classification problems. A classification problems typically involves a number of data samples, each of which is associated with a class (or label), and some features (or attributes). Given a previously unseen sample, the problem is to predict its class by looking at its features.

A support vector machine solves this problem by first building a model from a set of data samples with known classes (i.e., the training set), and use the model to predict classes of data samples that are of unknown classes. To evaluate the performance of an SVM, a testing set of data samples is only given with the features. The accuracy of the SVM can be defined as the percentage of the testing samples with correctly predicted class labels.

In essence, given the training set with class labels and features, a support vector machine treats all the features of a data sample as a point in a high dimensional space, and tries to construct hyperplanes to divide the space into partitions. Points with different class labels are separated while those with the same class label are kept in the same partition. This is often modeled as an optimization problem, and the goal of an SVM is to solve this problem. More details can be found in various previous work (e.g., Boser et al., 1992, and Cortes and Vapnik, 1995).

To build the model from the training set, the key is to choose the right kernel function and find its optimal parameters. Some of the most commonly supported kernel types include: (1) linear, (2) polynomial, (3) radial basis function (RBF), and (4) sigmoid. Each of the kernel functions may require a different set of parameters to be optimized.

Once a kernel function is chosen, the optimal parameters are selected using a k-fold cross-validation procedure. Typically, given a parameter k, the training set is randomly divided into k subsets of about the same size. Each time a new set of parameters is chosen, we use $k - 1$ subsets to train the SVM, and the last subset is used for testing. The accuracy is recorded, and the process is repeated for all possible combinations of $k - 1$ training subsets. The accuracies are then averaged and recorded as the accuracy of the selected parameters. This procedure is iterated for many possible combinations of parameters until a certain stop condition is satisfied. The parameters are optimized in this way to avoid over-training of the SVM, which yields good results on the training set but poor results for unseen testing data.

4. Proposed Method

4.1. Assumptions and Models

We assume that each time we are given one fragment for classification, and the result is independent of any other fragments. In reality, this memoryless model is clearly an over simplification, since neighboring memory fragments are likely to be of the same type, where neighboring may mean physical proximity or logical distance, e.g., consecutive memory address locations. However, such likelihood is often difficult to exploit since there lacks statistical models, for instance, for real world file systems. When such statistical models do exist, they can be applied in combination with memoryless data fragment classification techniques. Hence, we choose to focus on the memoryless model in this paper.

We consider n distinct types of data, where no type is a sub-type of another. In other words, given a data fragment, it should unambiguously belong to exactly one of the n types. For example, we can consider two types: JPEG and MP3, which are mutually exclusive1, but we cannot have TIFF and JPEG, since the former is a container type and may contain a JPEG image.

Given a data fragment and n distinct types, there are two kinds of questions we can ask about the type of the data fragment. The first is binary questions. For example, if

1 Strictly speaking JPEG can also be used as a container format, but such usage is rare. Hence we only consider commonly seen baseline and progressive JPEG formats.

we consider only two types A and B at a time, we can ask if the fragment is of type A or type B. We call these binary classification problems. These questions make sense in some scenarios. For example, if the only type of data we are interested in is JPEG images, we can ask if the fragment is JPEG or non-JPEG. If there are only two types to consider (e.g., JPEG and MP3), the question may be simply if the fragment is JPEG or MP3.

Another more general kind of questions are those that require to identify the given data fragment as one out of n types, where n > 2. This is often referred to as multi-class classification problems. In general the answer to this kind of questions cannot be obtained reliably or efficiently by asking binary questions, and hence they require a different treatment.

4.2. Feature Representation

We let f be a data fragment of ℓ bytes. The size ℓ is a multiple of the sector size, which is 512 in a typical file system. For example, ℓ may be 4096, a typical size of a cluster or a memory page in an operating system with paged memory. We treat each byte as an unsigned integer, with values from 0 to 255.

The feature vector vf of a data fragment f is defined as the sequence vf = <p0, p1, · · · , p255>, where pi is the frequency at which byte value i occurs in f. For example, if byte 0x00 occurs c times in a fragment of size ℓ, we compute p0 = c/ℓ. In other words, each feature vector represents the histogram of the data bytes.

When an SVM is applied, each data fragment (i.e., each feature vector) is considered as a point in a 256 dimensional space or mapped to a higher dimensional space. The objective is to construct hyperplanes to partition the space according to the different types and put these points into appropriate partitions in the space that represent the different types.

4.3. Training and Prediction

Support vector machines are a group of supervised machine learning algorithms that need to be trained before it can be used for classification. To train an SVM, we firstly gather data fragments from files with known types, and then we feed the feature vectors of the data fragments and their type information to the SVM.

As we mentioned earlier, the most commonly supported kernel types include: (1) linear, (2) polynomial, (3) radial basis function (RBF), and (4) sigmoid, and each of the kernel functions requires a different set of parameters to be optimized. Hence we first need to choose one of the kernel functions and then search for optimal parameters using cross validation techniques. In the end we determine the set of parameters to be used for the SVM and build corresponding trained models.

With the trained models and optimal parameters, we can ask the SVM to predict the type of a new data fragment. There are two types of errors that can occur with respect to a type A. The false-positive α of the SVM is the probability that a data fragment that is not of type A is classified as so, and conversely the false-negative β of the SVM is the probability that a data fragment of type A is not classified as so. The detection rate is $1 - \beta$ with respect to type A. In case of multiple file types, we can use the accuracy of the SVM as the measure, which is the probability that a given data fragment is wrongly classified.

5. Evaluation

5.1. Training and Testing Datasets

To create the training dataset, we collected 800 JPEG images, 800 MP3 music files, 800 PDF documents and 800 dynamic link library files (DLLs). Note that files with these types are usually of high entropy and the histograms of these file types are very similar. It is also noted that PDF and JPEG files are not exactly mutually exclusive since a PDF document may contain JPEG images as part of its data, which may affect the accuracy of our results. Nevertheless, most of the PDF documents we collected contain mainly texture content, and graphical content is not specifically excluded from the PDF documents.

For each file, we firstly divide it into fragments of 4096 bytes each, since this is the most common size for a cluster in modern file systems. We then remove the first and the last fragment from the dataset.

The reason for the removal of the first and the last fragments are twofold. On one hand, those fragments are easily identifiable using keywords, since there is typically a header and footer that mark the beginning and end of the file. On the other hand, these fragments usually contain data that is not part of the main data stream of the file. For example, the "header" portion of a JPEG image may contain textual content, and the bytes beyond the footer are the file slack, which may be of arbitrary values.

Note that our approach is slightly different from that is used in the Oscar method, in that we do not start specifically from the "main body" of the file (e.g., from the SOF marker of the JPEG images), since in practice it is very unlikely a fragment would begin exactly at the SOF marker.

We then compute the training feature vectors from the remaining parts of the training files by counting the frequencies at which each of the byte values occur in each file. We construct one feature vector from all the remaining fragments of each file in the training set.

In addition to the training data, we further collected 80 JPEG images, 80MP3 music files, 80 DLLs and 80 PDF documents for testing. The testing files are similarly divided into fragments of 4096 bytes with the first and last fragments removed.

We then compute testing feature vectors from the testing files. However, instead of computing one vector per file, we compute one vector per data fragment, since in practice we would not have the entire file at the time of classification.

5.2. SVM Training and Testing

To evaluate our classification method, we chose an SVM software package called libsvm (Chang and Lin, 2001), which is able to solve both binary and multi-class classification problems. This software is written in C++, and is easily portable on many platforms. We use the latest version 2.9 on Windows platform for our experiments. Libsvm supports the four kernel types we mentioned in Section 3. We consider two different types of classifications, namely, binary and multi-class classifications.

For binary classifications, we mainly focus on distinguishing one particular file type from other file types. To illustrate the effectiveness of our method, we test JPEG fragments against data fragments from files in DLL, MP3 and PDF formats, which are all of high entropies.

For multi-class classifications, we consider two cases. In the first case, we put the fragments from the files in JPEG, DLL and MP3 formats together and try to place the testing fragments into these three classes. In the second case, we add the fragments from the PDF files.

For each experiment, after choosing the training and testing datasets, an important next step is to linearly scale the feature vectors of the training data such that the values of each feature component fall within the range of $[-1, 1]$. The parameters for the scaling is saved and used to scale the corresponding feature components in the testing dataset.

For each case, we perform parameter optimization for the SVM through the following steps.

- Choose the kernel type.
- Choose a set of parameters according to the kernel type and previous results.
- Perform a 5-fold cross validation.
- Record the resulting accuracy.
- Check the completion condition and repeat from Step 2 if necessary.
- Repeat from Step 1 until all supported kernel types are tested.
- Note that Step 2 and 5 are specific to the kernel types and we omit the details here.

After the SVM is trained, we test the accuracy of the SVM by using data fragments extracted from the testing dataset. In each test case, we use 200 fragments of size 4096 bytes from each file type.

6. Results

The experiment results of binary classification are obtained using the JPEG image fragments against fragments in DLL, PDF and MP3 formats. The results are summarized in Table 1.

We observe that the SVM performs well when the kernel type is chosen correctly. In particular, linear kernel works best for 2 of the 3 file types, and works reasonably well for all 3 types.

For multi-class classification, we present the results in Table 2.

	DLL	**PDF**	**MP3**
Linear	98.25%	81.13%	89.13%
Polynomial	97.63%	49.44%	61.38%
RBF	96.94%	91.56%	44.25%
Sigmoid	96.44%	87.69%	69.19%

Table 1: Binary classification results on JPEG images vs. other formats

	3 classes (JPEG, DLL and MP3)	**4 classes (including PDF)**
Linear	87.17%	81.50%
Polynomial	71.00%	81.00%
RBF	76.33%	59.63%
Sigmoid	70.33%	52.50%

Table 2: Multi-class classification results on JPEG, DLL, MP3 and PDF fragments

From Table 2 we can see that, in general, the classification accuracy drops as the number of classes increases. Part of this performance drop may be due to the overlap between JPEG and PDF files. Nevertheless, the linear kernel SVM maintains high accuracies for multiple file types for these high entropy data fragments.

7. Conclusions

In this paper we studied the problem of the file type classification of evidence data fragments in the absence of header and file system information.

Previous statistical approaches are mainly based on computing the statistical distance between a given data fragment and known file types. A number of different statistical features have been used in previous work, including the entropy of the data, and the frequencies at which each byte value occurs in the data. These statistical methods are often used in conjunction with other heuristics. For example, certain patterns (keywords) are known to appear in certain file types more frequently, or would never appear in certain file types.

Although these previous techniques can achieve certain level of accuracy in some cases, it is known that good tools to reliably handle high entropy file types, such as multimedia files, compressed archives and executable programs, are lacking.

Recently, there have been attempts to solve the problem with machine learning techniques such as the Fisher linear discriminant. Despite the improved performance over previous methods, the classification system becomes complex and inefficient.

We proposed to utilize support vector machines, which are very powerful supervised learning algorithms that have been intensively studied in recent years. To maintain the simplicity of the classification system and achieve high efficiency, we employed a simple feature vector space, namely the byte frequencies, and trained the SVM with large amount of data and performed parameter optimization to achieve high accuracy.

Compared with the previous methods, we can achieve a high accuracy by using only a very simple statistics, the byte frequencies, which makes our scheme efficiently implementable and robust. We showed that, when trained with properly chosen parameters, SVM can be very powerful in differentiating data fragments from different types of files, even when no header or structure is available in the fragment.

8. References

Boser, B.E., Guyon, I., and Vapnik, V. (1992). “A training algorithm for optimal margin classifiers”, Proceedings of the Fifth AnnualWorkshop on Computational Learning Theory, pp 144–152. ACM Press.

Calhoun, W.C. and Coles, D. (2008). “Predicting the types of file fragments”, Proceedings of the 8th Digital Forensics Research Conference (DFRWS), volume 5 supp. 1 of Digital Investigation, pp S14–S20.

Chang, C.C. and Lin, C.-J. (2001). “LIBSVM: a library for support vector machines”. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm, (Accessed 31 March 2010).

Cortes, C. and Vapnik, V. (1995). “Support-vector networks”, Machine Learning, Volume 20, Number 3, pp 273-297. Springer Netherlands.

Damashek, M. (1995). “Gauging similarity with n-grams: Language-independent categorization of text”. Science, Volume 267, Number 5199, pp 843–848.

Erbacher, R.F. and Mulholland, J. (2007). “Identification and localization of data types within large-scale file systems”, Proceedings of the 2nd International Workshop on Systematic Approaches to Digital Forensic Engineering, pp 55–70.

Hall, G.A. and Davis, W.P. (2006). “Sliding window measurement for file type identification”, Technical report, Computer Forensics and Intrusion Analysis Group, ManTech Security and Mission Assurance.

Karresand, M. and Shahmehri, N. (2006). “File type identification of data fragments by their binary structure”, Proceedings of the 7th IEEE Information Assurance Workshop, pp 140–147.

Karresand, M. and Shahmehri, N. (2006). “Oscar - file type identification of binary data in disk clusters and RAM pages”, Security and Privacy in Dynamic Environments, Volume 201 of IFIP International Federation for Information Processing, pp 413–424. Springer Boston.

McDaniel, M. and Heydari, M.H. (2003). “Content based file type detection algorithms”, Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS’03). IEEE Computer Society.

Moody, S.J. and Erbacher, R.F. (2008). "SADI - statistical analysis for data type identification", Proceedings of the 3rd IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, pp 41–54. IEEE Computer Society.

Veenman, C.J. (2007). "Statistical disk cluster classification for file carving", Proceedings of the IEEE 3rd International Symposium on Information Assurance and Security, pp 393–398. IEEE Computer Society.

Li, W.-J., Wang, K., Stolfo, S.J., and Herzog, B. (2005). "Fileprints: Identifying file types by n-gram analysis", Proceedings of the 6th IEEE Information Assurance Workshop, pp 64–71.