Chapter 5

CONTEXT-BASED FILE BLOCK CLASSIFICATION

Luigi Sportiello and Stefano Zanero

Abstract Because files are typically stored as sequences of data blocks, the file carving process in digital forensics involves the identification and collocation of the original blocks of files. Current file carving techniques that use the signatures of file headers and footers could be improved by first classifying each data block in the storage media as belonging to a given file type. Unfortunately, file block classification techniques tend to have low accuracy. One reason is that they do not account for compound files that contain subcomponents encoded as different data types. This paper presents a context-based classification approach that accounts for compound files and improves on block-by-block classification schemes by exploiting the contiguity of file blocks belonging to the same file on the storage media.

Keywords: File carving, file block classification

1. Introduction

An important task in digital forensics is the retrieval of deleted files from storage media. Since files are typically stored as sequences of data blocks, the retrieval process involves the identification and collocation of the original blocks of each file. This is often performed using file system structures, which due to the persistence of file system metadata, still point to deleted data. When old data has to be retrieved or when the file system has extensive damage, it is necessary to use a "file carving" technique that reconstructs files based on their content. This is usually performed by relying on the signatures of known file headers and footers to detect the beginning and the end of each file on the storage media [10]. Of course, this creates problems when reassembling fragmented files in which blocks belonging to different files are interleaved [5].

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics VIII, IFIP AICT 383, pp. 67–82, 2012.

To perform file carving without relying on headers and contiguity, it is useful to classify blocks according to their file types based entirely on their content. Another application of file block classification is the detection of data hidden in locations that are not pointed to by the file system or residual data (e.g., in memory dumps or swap files and temporary files). A review of file carving techniques [9] underscores the importance of block classification in creating novel file carving solutions.

When using a classifier to perform file block classification, it is important to have a high detection rate because missing blocks may compromise file reconstruction. Also, it is necessary to have a low false positive error rate to reduce the computational complexity of file recovery.

However, existing classifiers [4, 12, 13] exhibit two problems. First, classification performance is far from perfect: false positives and false negatives are present, and they may hinder the reconstruction process. Thus, an improvement of the block-by-block classification approach is required. Second, some file types (e.g., `doc` and `pdf`) are inherently "compound" in nature, meaning that they may contain data encoded in other file type formats (e.g., an image embedded in a `pdf` file); this must be taken into consideration when creating the classifier.

This paper first demonstrates the impact that compound files have on a statistical block classification approach; this underscores the importance of considering compound files when designing and testing a statistical block classification approach. Next, a context-based classification approach is proposed that improves on block-by-block classification schemes of compound files by exploiting the contiguity of file blocks belonging to the same file on the storage media.

2. Related Work

Two primary approaches for classifying file blocks into their original file types are: (i) using a distance measure between a given input block and each reference model/sample; and (ii) applying machine learning techniques to create an appropriate classifier.

A distance-based approach [7, 8] performs the classification based on the frequencies of byte values and the differences between the values of consecutive bytes in a block. A set of files of each file type is used to compute the frequency model. If the distance between the frequencies of an unclassified block and one of the models is below a threshold, then the block is associated with the corresponding file type. Some solutions (e.g., [1, 2]) measure the distance between a pair of blocks by comparing the compression of the two individual blocks with the compression of their concatenation. In this case, a block is classified by computing its distance

from sample blocks representing different file types and associating the block with the file type of the closest sample block.

A machine learning approach uses statistical values as a set of input features in a classification algorithm. The algorithm is then used to classify new samples based on the learned model. A learning algorithm typically employs Fisher's linear discriminant [4, 13] or a support vector machine (SVM) [12]. Since multiple file types exist, the problem can be categorized as multi-class classification. Binary classifiers can be used to distinguish between the multiple file types by generating several one-to-one classifiers [4] that distinguish individual file types from each other. Alternatively, one-to-many classifiers [12] can be created to separate a single file type from other file types. Two variations of multiple classifiers for identifying blocks have been proposed: one directly assigns a file type to a given input block; the other discerns specific file type blocks in a block set [13].

3. File Block Classifiers

The purpose of file block classification is to assign a file type to a file block based only on its content. This work addresses the problem of detecting all the blocks belonging to a specified target file type in a block set (e.g., disk image). This is structured as a one-to-many classification problem in which a binary classifier for each file type is trained to discern blocks of the target type from among all the other file types.

A support vector machine (SVM) [3] is used as the classifier. An SVM performs binary classification based on a training set $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, l$, with each sample $\mathbf{x}_i$ represented by n attributes (features) in the space $\mathbb{R}^n$ and labeled with a class $y_i \in \{1, -1\}$. The SVM obtains a maximally separating hyperplane of the form $\mathbf{w} \cdot \mathbf{x} + b = 0$ by solving the optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

Such a hyperplane linearly separates the space $\mathbb{R}^n$ into two regions representing the two classes $\{1, -1\}$. A new data sample is assigned to a class according to the side of the hyperplane where the sample lies. Since the training samples $\mathbf{x}_i$ may not be linearly separable in $\mathbb{R}^n$, to improve the classification, they are mapped to a higher dimensional space by the

function ϕ. The linear separation is achieved in the higher dimensional space, resulting in a non-linear separation in the original space $\mathbb{R}^n$.

The mapping uses a suitable kernel function $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. In our approach, we use the RBF kernel function $K(x, y) = e^{\gamma \|x-y\|^2}, \gamma > 0$, a popular choice that is suitable for many problems [6]. Thus, the classifier parameters are: γ (kernel function parameter) and C (misclassification penalty parameter).

In a classification process, it is important to represent the samples using a set of features that highlight the differences between items from different classes. Our classification technique uses the features defined by Sportiello and Zanero [12]. These features include:

- **Byte Frequency Distribution (BFD):** This feature is based on the frequencies f_v of each possible byte value $v \in \{0, \ldots, 255\}$ in the block.
- **Rate of Change (RoC):** This feature is based on the frequencies of the differences between two consecutive bytes b_i and b_{i+1} in a block (i.e., distribution of $b_i - b_{i+1}$).
- **Word Frequency Distribution (WFD):** This feature is similar to BFD, but it considers a block as a sequence of 16-bit words and computes their frequency values.
- **Mean Byte Value:** This feature interprets a block as a sequence of bytes and computes the average value.
- **Entropy:** This feature interprets a block as a sequence of bytes and computes the entropy value.
- **Lempel-Ziv Complexity:** This feature interprets a block as a binary stream and computes the Lempel-Ziv complexity.

Various combinations of these features were tested. The feature combinations that provided the most accurate classification results were used for file block classification.

3.1 Compound File Problem

Storage media may contain a variety of file types. Many of these file types encode data in a similar pattern throughout the length of the file. Good examples are image and video files. They are called "primitive" files because the blocks corresponding to these file types tend to present common features that can be exploited in identifying the blocks.

On the other hand, "compound" files have distinctive file structures in which data encoded in the form of primitive types may be embedded.

Such files are created, for example, when an image is embedded in a document file by a word processor or when a video is included in a presentation. In these cases, the file blocks corresponding to the file do not present uniform properties because the blocks related to the embedded data are statistically different from the other file blocks.

To perform file block classification in the presence of compound files, Roussev and Garfinkel [11] recommend classifying each block as a primitive type and then, during compound file recovery, use their internal structures to recover the complete file. Conducting this type of recovery requires the ability to distinguish blocks that constitute the basic structure of a compound file from the data blocks of other formats. This is, in fact, the focus of this paper.

Certain problems posed by compound files must be taken into consideration when creating models for block classification. In particular, with regard to a supervised learning approach, the training set should be prepared in a proper manner. To understand this, consider the training set for a given primitive file type. The training set would comprise examples and counterexamples. However, the training set would mislead the classifier if blocks from a compound file with embedded data that is encoded according to the target type format were to be mistakenly included among the counterexamples. In fact, the training set would comprise blocks of the target file type data that are labeled as both target and non-target, inducing poor classification performance. For this reason, when compound files are included in a training set, it is prudent to use compound file types that do not contain embedded data.

3.2 Experimental Setup

Classifiers were constructed using the methodology described in [12], except for differences in data set preparation to account for the compound file problem. The experiments used the same data set of randomly downloaded files as in [12], which includes `bmp`, `doc`, `exe`, `gif`, `jpg`, `mp3`, `odt` and `pdf` files. Because `doc`, `exe`, `odt` and `pdf` are compound file types, the corresponding files were inspected for embedded data. In constructing the file block classifiers, all files containing embedded data were replaced with files without embedded data that were randomly downloaded from the Internet using the same approach as in [12].

The collected files were decomposed into 512-byte blocks, yielding an average of 28,000 blocks per file type. The length of 512 bytes was selected because it is the smallest block size that is commonly used to manage storage media. Also, as described in [12], a smaller block size

Table 1. Feature set used to represent file blocks.

Feature	Description
Entropy	File block entropy
Complexity	File block Lempel-Ziv complexity
BFD	Frequency of byte values in the file block
Entropy-Complexity-BFD	Concatenation of entropy, complexity and BFD
RoC	Frequency of differences between two consecutive byte values in the file block

renders the classification task more difficult; therefore, using the smallest value yields conservative performance results.

For each file type, we constructed an SVM classifier to detect the corresponding blocks. For each classifier, it was necessary to set the relative values of the parameters γ and C, and to select the features for block representation. To identify the best feature-parameter combination for each file type, a seven-fold cross validation was conducted, which split the data set into a training set and test set containing 2,000 and 8,000 blocks, respectively. In each training set, half of the blocks were of the target file type, while the remaining blocks uniformly represented the other file types. For each file type we trained and tested a series of classifiers by varying the parameters $\gamma \in \{2^{-15}, 2^{-13}, 2^{-11}, \ldots, 2^{5}, 2^{7}\}$ and $C \in \{2^{-5}, 2^{-3}, 2^{-1}, \ldots, 2^{13}, 2^{15}\}$, and attempting combinations of all the features as well as reduced versions (e.g., BFD related only to ASCII byte values) and concatenations (e.g., Entropy-Complexity-BFD) [12]. The feature set selected was the combination that maximized the function $0.5 \cdot \text{TP} + 0.5 \cdot (1 - \text{FP})$, where TP and FP denote the true positive and false positive error rates, respectively. After the best combinations for each file type were identified, they were used to create a final set of classifiers that relied on the entire block collection, with training sets of 28,000 blocks and test sets of 112,000 blocks.

3.3 Experimental Results

Table 1 lists the features used to test file type classification. Table 2 presents the SVM parameters used for each file type and the final classification results. The concatenation Entropy-Complexity-BFD proved to be the most effective feature representation for all the file types, except for `bmp`, for which RoC is marginally better. Thus, this file block representation can work well for most different file types.

Because the compound files were removed, the classification results are better than those presented in [12]. This is particularly evident for

Table 2. File block classification by SVMs (no embedded data in compound files).

	Feature	γ, C	TP	FP
bmp	RoC	$2^1, 2^9$	99.6	1.7
doc	Entropy-Complexity-BFD	$2^3, 2^3$	91.0	2.4
exe	Entropy-Complexity-BFD	$2^1, 2^5$	87.1	0.1
gif	Entropy-Complexity-BFD	$2^5, 2^1$	95.5	3.9
jpg	Entropy-Complexity-BFD	$2^5, 2^1$	96.4	3.9
mp3	Entropy-Complexity-BFD	$2^5, 2^1$	96.9	2.8
odt	Entropy-Complexity-BFD	$2^5, 2^1$	96.8	16.7
pdf	Entropy-Complexity-BFD	$2^3, 2^1$	94.4	19.8
Average			94.7	6.4

	FP per File Type							
	bmp	**doc**	**exe**	**gif**	**jpg**	**mp3**	**odt**	**pdf**
bmp	–	8.0	3.0	0.1	0.2	0.2	0.1	0.3
doc	10.4	–	5.2	0.2	0.2	0.7	0.1	0.3
exe	1.3	3.6	–	0.3	0.2	0.1	0.0	0.3
gif	0.7	5.3	1.7	–	3.2	2.7	6.8	6.9
jpg	0.4	0.5	1.2	2.1	–	6.0	10.7	6.2
mp3	0.9	0.6	2.9	2.6	5.3	–	4.8	2.5
odt	0.4	6.6	8.5	12.3	20.4	10.6	–	57.7
pdf	0.5	6.2	7.7	16.1	16.9	7.2	84.0	–

the FP rate of the `doc` classifier, which decreased from 19.8% in [12] to the current value to 2.4%. Also, the FP rates against `gif` and `jpg` (two common types of embedded data in `doc` files) reduced by 6% and 28%, respectively. Likewise, the elimination of compound files improved the `gif` and `jpg` classifiers; their specific FP rates against `doc` files decreased by 3% and 15%, respectively.

The classifiers support data recovery of primitive file types (e.g., `bmp` and `jpg`). However, because the recovery of compound files (e.g., `doc` and `pdf`) requires the handling of embedded data, the classifiers provide limited support for compound file types (techniques such as those described in [11] should be used for these file types).

4. Context-Based Block Classification

Most of the files of interest in a forensic recovery process (e.g., documents, images and videos) are typically not small in size and, thus, span multiple blocks [5]. Modern file systems tend to reduce file fragmentation, meaning that blocks belonging to the same file are stored in contiguous locations to the extent possible. When fragmentation occurs,

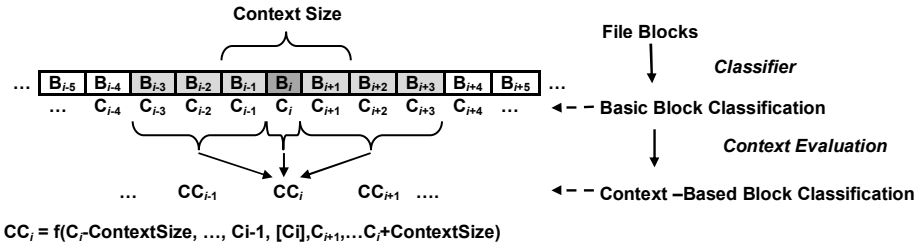

Figure 1. Context-based file block classification.

the most common scenario is bi-fragmentation, where a file is stored as two series of contiguous blocks far from each other on the media [5]. Thus, it follows that blocks are typically surrounded by other blocks of the same file (and the same file type), except for those at the beginning and at the end of a file or its fragments.

Even if the error rates of a block classifier are low, the misclassifications that occur are multiplied when dealing with many terabytes of data; this can significantly impact the final results. Therefore, it is necessary to improve the precision of block classification. Also, when there is insufficient information to make a correct decision, it is preferable to mark a block as "not-classified" instead of giving it an incorrect classification. This helps reduce errors in the processes that use the classification information [11].

Our approach exploits file block contiguity. When classifying a block B_i, we rely on the classifications of its neighboring blocks because they tend to belong to the same file. We call such blocks the "context" of B_i (Figure 1). We use the *ContextSize* parameter to denote the number of neighbor blocks to be considered on each side of B_i, and combine the classifications of the blocks with the classification C_i of the block itself. This yields a "context-based" classification $CC_i = f(C_{i-ContextSize}, \ldots, C_{i-1}, [C_i], C_{i+1}, \ldots, C_{i+ContextSize})$. The idea is that if a good, but not perfect, classifier is available, a more robust evaluation may be achieved by relying on a series of classifications (context).

4.1 Block Context Evaluation

The block context evaluation classification model is binary: for a given target file type, for each generic block B_i, the classifier outputs classification $C_i \in \{-1, 1\}$, where 1 means the block is of the target type and -1 means it is not.

The context-based classification CC_i for a block B_i is performed according to Algorithm 1. The classification of the context blocks preced-

Algorithm 1 : Context-based classification CC_i of a block B_i

```
LeftClassifications={C_{i-1}, C_{i-2}, ..., C_{i-ContextSize}}
RightClassifications={C_{i+1}, C_{i+2}, ..., C_{i+ContextSize}}
LeftEvaluation=ContextEvaluation(LeftClassifications)
RightEvaluation=ContextEvaluation(RightClassifications)
if LeftEvaluation>0 && RightEvaluation>0 then
   if Ignore C_i then
      CC_i = 1
   else if Consider C_i then
      if C_i>0 then
         CC_i = 1
      else
         CC_i = NC [Not-Classified]
      end if
   end if
else if LeftEvaluation<0 && RightEvaluation<0 then
   if Ignore C_i then
      CC_i = -1
   else if Consider C_i then
      if C_i<0 then
         CC_i = -1
      else
         CC_i = NC [Not-Classified]
      end if
   end if
else if LeftEvaluation==0 && RightEvaluation!=0 then
   if C_i>0 && RightEvaluation>0 then
      CC_i = 1
   else if C_i<0 && RightEvaluation<0 then
      CC_i = -1
   else
      CC_i = NC [Not-Classified]
   end if
else if LeftEvaluation!=0 && RightEvaluation==0 then
   if LeftEvaluation>0 && C_i>0 then
      CC_i = 1
   else if LeftEvaluation<0 && C_i<0 then
      CC_i = -1
   else
      CC_i = NC [Not-Classified]
   end if
else
   CC_i = NC [Not-Classified]
end if
```

ing B_i (i.e., "left" context) is performed using the *ContextEvaluation* function, which returns a value from $\{-1, 1\}$ expressing the number of considered blocks that correspond to the target or non-target (positive

Table 3. Context evaluation functions.

	$\boldsymbol{ContextEvaluation(C_1, C_2, ..., C_n)}$
Uniform	$\frac{1}{n}\sum_{i=1}^{n} C_i$
Exponential	$\sum_{i=1}^{n} \frac{C_i e^{-(i-\frac{1}{2})/n}}{weight}$ with $weight = \sum_{i=1}^{n} e^{-(i-\frac{1}{2})/n}$
Linear	$\sum_{i=1}^{n} \frac{C_i(1-\frac{1}{n}(i-\frac{1}{2}))}{weight}$ with $weight = \sum_{i=1}^{n} 1 - \frac{1}{n}(i-\frac{1}{2})$

or negative value, respectively). The context blocks following B_i ("right" context) are evaluated in a similar manner.

Our experiments compared two variants of the algorithm. The first variant ignores the classification C_i of B_i and relies only on the context. The second variant includes C_i with the context during classification. The final classification CC_i is deemed a "target" if both the evaluations of the context have positive values (this is interpreted as B_i being in the middle of the target file). Similarly, if the two evaluations present negative values, then the block is deemed a "non-target." In the second variant, where C_i is taken into account, if its value agrees with the two context evaluations, then the final classification is the same as that obtained using the first variant; otherwise, the output is "not-classified" because of the mismatch in the available information.

Some special cases are considered. If one of the two context evaluations returns zero (perhaps because the context is placed over the blocks of two different contiguous files), then the block classification C_i is compared with the non-null evaluation; if they agree, then the block is classified accordingly, otherwise, "not-classified" is returned. Also, if the two evaluations disagree or are both equal to zero, then the block is labeled as not-classified. This is because the available information is inconsistent.

Note that the left context is null for the first block of a disk image. It grows in size for each following block until it reaches the defined *ContextSize*. Symmetric behavior occurs at the end of the disk image, where $RightEvaluation = 0$ for the last block.

Table 3 shows the three *ContextEvaluation* functions that were tested. The *Uniform* function computes the average of the classifications of all the blocks in a context without any weights. The *Exponential* and *Linear* functions give higher weights to classifications of blocks that are closer to B_i. The idea is that these blocks are more likely to be part of the same file as B_i, so their support for the final decision is higher.

4.2 Experimental Setup

The second set of experiments focused on improving the detection of `gif` files using context-based classification. The `gif` files were chosen because they are a primitive file type whose classifier, as shown in Table 2, performs well. In fact, we show that a good classifier can be used as a basis in our context-based algorithm to achieve better classification. Note, however, that the concept is general although our experiment uses SVM classifiers as described above.

The experiments used a new data set comprising 1.5 MB of files downloaded from the Internet, each file was roughly 100 KB in size. All the collected files were divided into 512-byte block sequences. Four "disk image" scenarios were created:

- **Scenario 1:** The files were randomly concatenated to create a disk image without fragmentation.
- **Scenario 2:** The files were split into two equal fragments that were randomly concatenated to create a bi-fragmented disk image.
- **Scenario 3:** The files were split into three equal fragments that were randomly concatenated to create a tri-fragmented disk image.
- **Scenario 4:** The files were split into ten equal fragments that were randomly concatenated to create a ten-fragmented disk image.

These scenarios were designed to test the context-based classification technique on common situations (corresponding to Scenarios 1-3) [5]. Scenario 4 corresponds to a possible, albeit unlikely, high fragmentation situation.

The experiments involved two stages. First, the `gif` classifier was used on all the blocks of a generated disk image to yield their basic classifications. Next, the context-based classification for each block was computed using the output of the `gif` classifier. The experiments compared the two variants of the algorithm that include and ignore the classification C_i of the block itself. Also, they explored $ContextSize$ parameter values of 3, 5, 8 and 10.

4.3 Experimental Results

Figures 2 and 3 present the true positive (TP), true negative (TN), false positive (FP) and false negative (FN) rates, along with the not-classified (NC) rates, since TP + FN $<$ 100% and TN + FP $<$ 100% due to the presence of NC blocks. Figure 2 shows the context-based file block classification results for the non-fragmented and bi-fragmented

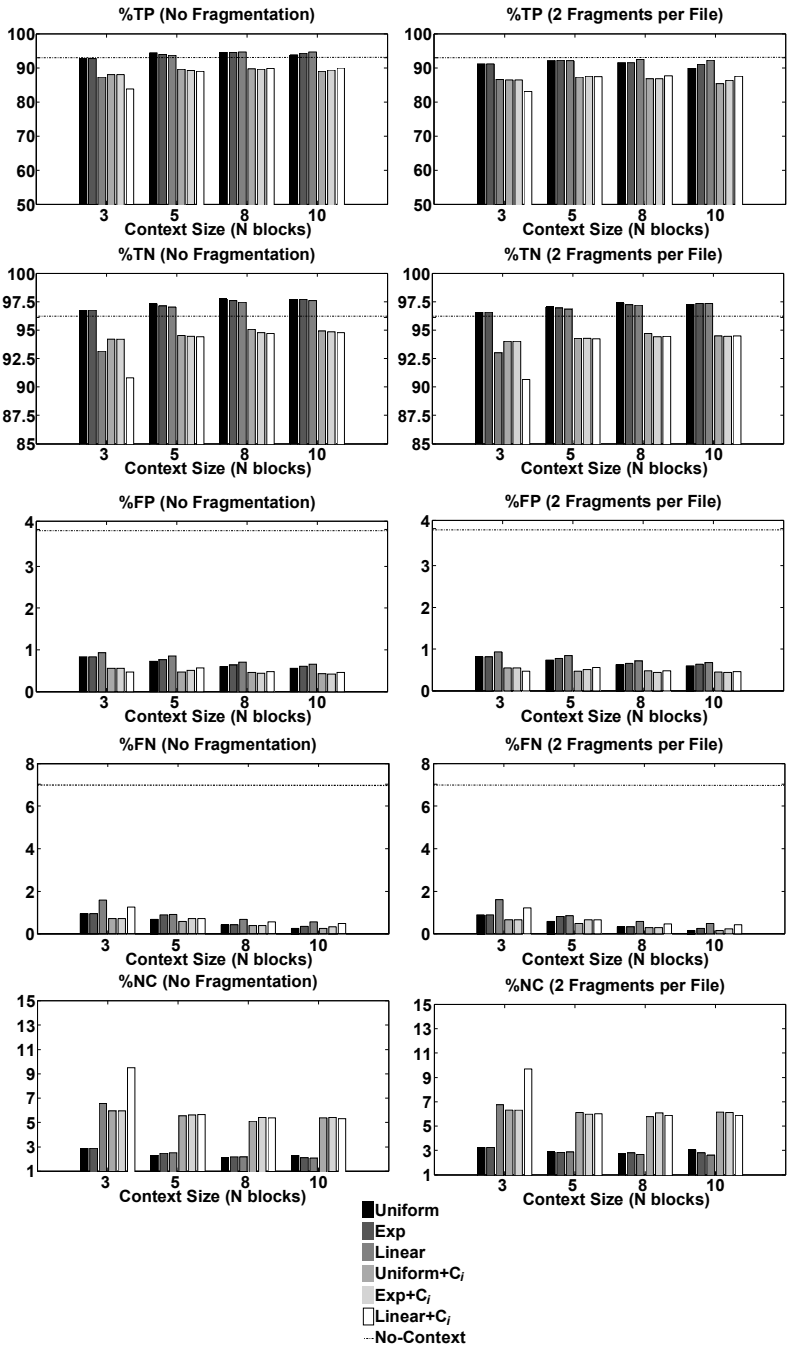

Figure 2. Context-based file block classification results.

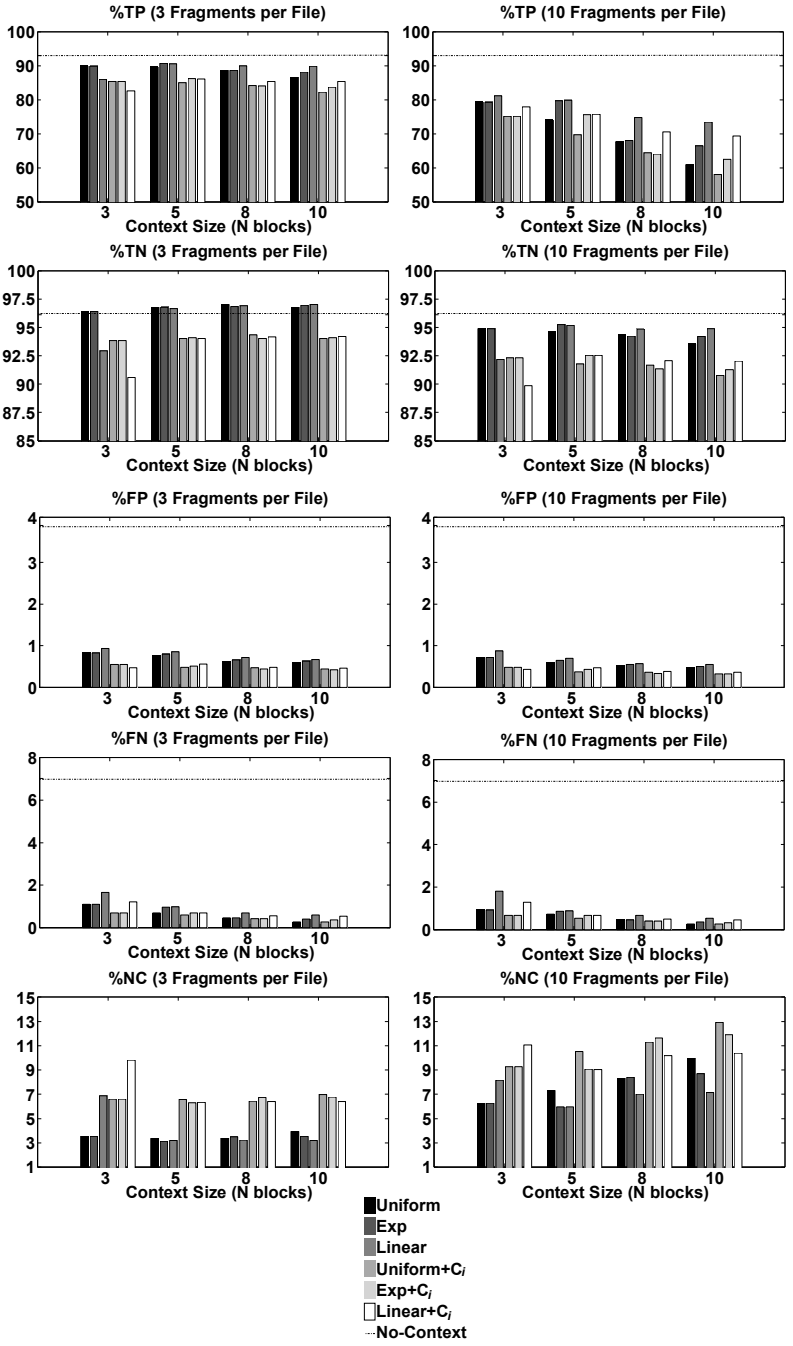

Figure 3. Context-based file block classification results.

file scenarios. Figure 3 shows the context-based file block classification results for the tri-fragmented and ten-fragmented file scenarios. The graphs also show the basic classification results, achieved using the `gif` classifier without a context as a baseline.

The first key result concerns the FP and FN rates. Both the error rates are strongly improved (i.e., decreased) compared with the basic classification for all the fragmentation scenarios and parameter values. For the first three scenarios, the TN rates are improved when the classification of the block itself is ignored. On the other hand, the TP rate decreases with increasing fragmentation, but increases in Scenario 1 (note that that non-fragmentation is the most likely condition for a file). These results can be explained by the presence of a long series of contiguous non-`gif` blocks that provide a wide base for the context-based classification of non-target blocks, while smaller block series related to `gif` files and their fragments are present to be exploited in target block identification. Also, roughly 3% of all blocks in Scenarios 1-3 are not-classified due to the `gif` blocks being incorrectly classified as target blocks. This is clearer in the graphs for Scenario 4, where a drop in the TP rate is correlated with an increase in the NC rate.

As expected, the classification performance decreases when the fragmentation rate increases. However, in context-block classification, the TP and TN rates tend to be converted to NC rather than to FP and FN, keeping the rates low and maintaining a lower classification error.

Increasing the *ContextSize* parameter helps reduce the FP and FN rates; the TN rate is also slightly increased, except that for the last high fragmented scenario. However, the TP rate does not seem to be positively affected, probably due to the absence of target block series long enough to be exploited by the enlarged context. A *ContextSize* value of five appears to represent a good trade-off in our experiments.

The results suggest that it is better to ignore the classification of the block itself when classifying a block. In fact, taking C_i into account negatively affects the TP and TN rates, leaving the FP and FN rates substantially similar, due to an increased NC rate.

The three *ContextEvaluation* functions exhibit comparable performance. The *Linear* function performs worse than the other two functions for a *ContextSize* value of three, probably because the short context causes the function to focus mainly on the two contiguous blocks of B_i. On the other hand, the *Linear* function outperforms the others in the high fragmentation scenario. In this scenario, it effectively reduces the effect of the non-target blocks in the context (a typical situation close to fragment boundaries), relying on the context blocks closer to the block under classification.

The classification approach tends to concentrate NC blocks mostly at the boundaries of files or fragments. A file carver could exploit this fact to identify block regions belonging to a specific file type and related to a file or fragment on the storage media, but with "faded" boundaries – correct classification in the middle of fragments with NC values near the ends. Then, the file carver could attempt to collate these block regions, varying their length in the range identified by the relative NC areas, until a complete file is recovered (the various combinations could be checked using a file validator [5]).

5. Conclusions

This paper makes two contributions to file block classification for forensic data carving applications. The first is a technique that improves the performance of single block file block classifiers – false positive and false negative error rates are reduced using training sets restricted to primitive file types. The second is a context-based classification methodology that exploits the spatial coherence of data, i.e., the contiguity of blocks related to a given file. This methodology improves block classification performance by covering misclassifications at the cost of introducing a limited number of non-classified blocks. The methodology is general and can be applied in conjunction with other content-based file block classification algorithms. Our future research will focus on this aspect and will also assess the impact of the methodology on other block classification strategies.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the European Commission.

Acknowledgements

This research was partially supported by the Prevention, Preparedness and Consequence Management of Terrorism and Other Security-Related Risks Program of the European Commission under Project i-Code: Real-Time Malicious Code Identification; and by the EU Seventh Framework Program (FP7/2007-2013) under Grant No. 257007 – SysSec.

References

[1] S. Axelsson, The normalized compression distance as a file fragment classifier, *Digital Investigation*, vol. 7(S), pp. S24–S31, 2010.

[2] S. Axelsson, Using normalized compression distance for classifying file fragments, *Proceedings of the Fifth International Conference on Availability, Reliability and Security*, pp. 641–646, 2010.

[3] C. Burges, A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, vol. 2(2), pp. 121–167, 1998.

[4] W. Calhoun and D. Coles, Predicting the types of file fragments, *Digital Investigation*, vol. 5(S), pp. S14–S20, 2008.

[5] S. Garfinkel, Carving contiguous and fragmented files with fast object validation, *Digital Investigation*, vol. 4(S), pp. S2–S12, 2007.

[6] C. Hsu, C. Chang and C. Lin, A Practical Guide to Support Vector Classification, Technical Report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2003.

[7] M. Karresand and N. Shahmehri, File type identification of data fragments by their binary structure, *Proceedings of the 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 Twenty-First International Information Security Conference*, pp. 413–424, 2006.

[9] A. Pal and N. Memon, The evolution of file carving, *IEEE Signal Processing*, vol. 26(2), pp. 59–71, 2009.

[10] G. Richard and V. Roussev, Scalpel: A frugal, high performance file carver, *Proceedings of the Fifth Digital Forensics Research Workshop*, 2005.

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

[12] L. Sportiello and S. Zanero, File block classification by support vector machines, *Proceedings of the Sixth International Conference on Availability, Reliability and Security*, pp. 307–312, 2011.

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