Chapter 1

CONTEXT-BASED FILE BLOCK CLASSIFICATION

Luigi Sportiello, Stefano Zanero
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milan, Italy
luigi.sportiello@mail.polimi.it, stefano.zanero@polimi.it

Abstract In computer forensics, carving is an important trick in the digital investigator's sleeve. Since files are typically stored as sequences of data blocks, the retrieval process basically consists of locating and appropriately collating together the original blocks of each file. Traditional file carving solutions, generally based on signatures of file headers and footers, could be improved by performing a classification of each data block in the storage media as belonging to a given file type. Unfortunately file block classification techniques tend to be far from perfect in terms of accuracy. For an improvement of the classification results the presence of compound files, i.e. files containing sub-portions that are encoded similarly to a different data type, must be taken into account during the classifier preparation. In this work, we demonstrate that this impacts heavily on the performance of file block classifiers. In addition, to generally improve the accuracy of classification, we propose a context-based classification architecture to improve block-by-block classification schemes, by exploiting the contiguity of file blocks belonging to the same file on storage media. The approach is completely general and can be easily applied to any content-based file block classification algorithm.

Keywords: File Block Classification; File Carving; Forensic Analysis

1. Introduction

Forensic applications of computer science are growing in importance, in parallel with the growth of the prevalence of cybercrime and of digital evidence in traditional crime. Within the toolkit of the digital forensic

analyst, a fundamental role is played by tools for the retrieval of deleted files from storage media. Since files are typically stored as sequences of data blocks, the retrieval process basically consists of locating and appropriately collating together the original blocks of each file.

While, due to persistence of file system metadata, this is often performed by using file system structures that still point at the content of deleted data, in case of retrieval of older data, or in case of extensive file system damage, this can be performed with a set of techniques known as "carving", i.e. approaches to reconstruct files based on their contents. This is usually performed by relying on signatures of known headers and footers of files [1], to detect the beginning and the end of each file on the storage media. Of course, this creates obvious challenges when reassembling fragmented files, where blocks belonging to different files may be interleaved; this has been shown [2] to be important for digital forensic.

In order to perform carving without relying only on headers and contiguity, or more in general to assist carving and improve resilience against fragmentation and intermission of files on disks, it may be useful to be able to classify blocks according to the file types they originally belong to, relying exclusively on their content. Other possible applications of file block classification are contrasting anti-forensic techniques based on manipulation or disruption of file system metadata and file headers, as well as detection of data hidden in locations not pointed to by the file system or residual data, for instance in memory dumps or in swap and temporary files. A recent review of file carving techniques can be found in [3], where also a recent file carver which makes use of block classification techniques is presented. This remarks the importance of block classification in novel file carving solutions.

In [4], for instance, we showed how to perform file block classification by using Support Vector Machines (SVMs). In particular, we computed a classification model for each considered file type, and then exploited them for the identification of the corresponding file blocks. Computed classifiers should present a high detection rate, since missing some blocks may compromise the reconstruction of files, while a low false positive rate may help in keeping the recovery computational complexity low. However, the classifiers presented in the aforementioned work, in spite of offering a remarkable accuracy, exhibit two challenging problems, which are similarly present in many other approaches in literature.

First, classification performance, even if pretty good, is far from perfect: false positives and false negatives are present, and they may hinder or block altogether the reconstruction process, since a large amount of blocks may need to be analyzed. Thus, a further improvement in block-by-block classification processes is needed. Additionally, some file types

(e.g, `doc`, `pdf`) are inherently *compound*, meaning that they may contain data encoded according to other file type formats (e.g., an image embedded in a `pdf` file), and this should be taken into consideration during the classifier preparation.

In this work, using our own previous work as a running example, we first demonstrate the impact of compound files on a statistical block classification approach, showing that this aspect has to be taken in careful consideration both when designing and when testing such approaches. Then, we propose a new and general architecture to improve block-by-block classification schemes by exploiting the contiguity of file blocks belonging to the same file on storage media. We will call this approach a "context-based classification" in the following. It is important to note that, while the present paper builds on our own proposed classifier as a working example, the approach we propose is completely general and can be easily applied to any content-based file block classification algorithm.

The rest of the paper is structured as follows: Section 1.2 summarizes related work in the area, helping to place our work in context. In Section 1.3 we present our approach for the computation of SVM-based file block classifiers and its impact against the problem of compound files. In Section 1.4 we propose our context-based approach and validate its effectiveness by a comprehensive experimental evaluation. Finally, Section 1.5 allows us to draw some conclusions and outline future research perspectives.

2. Related Work

Type classification of a given file relying on its content has been developed in [5–7]. In this paper, we do not address the classification of a file as a whole, but rather we focus on the classification of single blocks according to the file type of the original file they belong to. In the last few years, different solutions have been experimented for the classification of file blocks into their original file type, and in particular two main approaches have been explored to solve the problem: computing a distance between a given input block and some reference models/samples, or adopting machine learning techniques to create appropriate classifiers.

Focusing on the first class of solutions, the authors of [8, 9] propose to perform the classification by relying on the frequencies of byte values and on the differences between values of consecutive bytes in a block. A set of files of each given 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 predefined threshold, the block is associated to that file type. The work presented in [10, 11] measures the

distance between a pair of blocks by comparing the compression of the two separate blocks with the compression of their concatenation: the better the compression of the concatenation, compared with the distinct ones, the more similar the blocks. Also in this case the block is classified by computing its distance from sample blocks representing different file types and associating it to the closest one.

With regard to the machine learning-based approaches, the authors of [12, 13] adopt a Fisher classifier, using several statistical values as a set of input features for representing the file blocks. Blocks of known file type are used as examples to compute the Fisher classifiers, which are then exploited to classify unknown blocks. The author of [12] propose two kinds of classifiers, one to directly assign a file type to a given input block, and the other to discern specific file type blocks in a block set. Differently, in [13] a pairwise classification problem is addressed, computing classifiers able to discern only between two specific file types. In [4] we proposed a block classification based on Support Vector Machines. For each considered file type, we generated a SVM classifier to discern its blocks against those of other file types. For each SVM, we showed how to properly select the best classifier parameters, and explored the feature selection problem for block representation.

3. File Block Classifiers

The purpose of file block classification is to assign a file type to a file block relying only on its content. This is a typical classification task, since a category has to be assigned to a given input item, and different solutions are available in the literature for it. In particular, we approach the classification problem using supervised learning, i.e., we exploit a labeled training set of data samples to construct models able to predict the category of unlabeled data samples.

In this work we address the problem of detecting all blocks belonging to a specified target file type in a block set (e.g., in a disk image). Formulated in this way, this is a binary classification problem, where we build a classifier per each target file type, training each of them with blocks from files of the target type and blocks from other kinds of files: the computed models have to be able to discern blocks of the target type against the others.

We adopt Support Vector Machines (SVMs) [14] as binary classifiers. In SVMs, on the base of a training set $(\mathbf{x}_i, y_i), i = 1, \ldots, l$, with each example $\mathbf{x}_i$ represented by n attributes (features) in the space $\mathbb{R}^n$ and labeled to a category $y_i \in \{1, -1\}$, a hyperplane of the form $\mathbf{w} \cdot \mathbf{x} + b = 0$

is computed by solving the following 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 $\mathbb{R}^n$ in two regions representing the two categories $\{1, -1\}$. A new data example is assigned to either category according to which side of the hyperplane it lies. Since the training examples $\mathbf{x}_i$ may not be linearly separable in $\mathbb{R}^n$, to improve the classification they are mapped into a higher dimensional space by the function ϕ, and the linear separation is achieved in the higher space, resulting in a non-linear separation in the original space $\mathbb{R}^n$. For such a mapping the user needs to select a suitable *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. We adopted the RBF kernel function $K(x, y) = e^{\gamma\|x-y\|^2}, \gamma > 0$, a very common choice since it is suitable in most cases [15]. Therefore, the parameters of our classifiers are: γ, a parameter of the kernel function; and C, a parameter which sets the penalty for misclassification.

In any classification process, it is particularly important to represent the samples using an appropriate set of features, suitable to highlight the differences among items of different categories. Feature selection is therefore a particularly delicate step. In this paper, we use the same set of features, and their combinations, presented in [4], in order to be able to compare the results. In particular, the features in use are:

- Byte Frequency Distribution (BFD), the frequencies f_v of each possible byte value $v = [0, 255]$ in the block;
- Rate of Change (RoC), the frequencies of the differences between two consecutive bytes b_i and b_{i+1} in the block (i.e. the distribution of $b_i - b_{i+1}$);
- Word Frequency Distribution (WFD), similar to BFD but considering a block as a sequence of 16 bit words and computing their frequency values;
- Mean Byte Value of the block, interpreted as a byte sequence
- Entropy of the block, interpreted as a byte sequence
- Lempel-Ziv complexity reading the block as a binary stream.

Different combinations of such features are tested in our classification process, and the ones providing the best classification results are used.

3.1 The Compound File Problem

In a storage media a wide range of file types may be present, each characterized by the type of data contained and by the way in which such data are organized. In particular, many file types present a peculiar and quite uniform way to encode the relative data along the whole file length: good examples are image and video files. These can be considered as a sort of *primitive* files. This means that the blocks composing such file types tend to present common features, which can be exploited to identify blocks belonging to them.

On the other hand, *compound* files also exist. They tend to present a basic and distinctive file structure, in which other data encoded in a different way (e.g. in a primitive type) may be embedded. This is for instance the case when an image is embedded in a document file by some word processor, or when a video is included in a presentation. In such cases, the file blocks belonging to the whole file do not present uniform properties anymore, because the blocks related to the embedded data look very different from the other blocks composing the file.

Compound files create issues for file block classification. In fact, blocks related to embedded data may completely look like blocks of other file types, and not resemble at all the blocks of the compound file they are contained in. In such a case, as suggested also in [16], it is advisable to classify such blocks as primitive types according to their encoding. Then, during a compound file recovery, relying on its internal structures, when the beginning of an embedded section is detected, blocks classified as the correct embedded type will be expected to appear. Thus, it could be useful to distinguish blocks that constitute the basic structure of a compound file against the data blocks of other formats.

Compound file issues have to be taken in particular consideration during computation of models for block classification. In particular, with regard to the supervised learning approach that we adopt, the training sets should be properly prepared. To understand why, it is sufficient to consider the training set for a given primitive file type. Such a training set comprises examples and counterexamples. What happens if we mistakenly include, among the counterexamples, blocks from a compound file embedding data which is actually encoded according to the target type format? Obviously, the training set will be misleading for the classifier. In fact, it will comprise blocks of the target file type data labeled as both target and non-target, inducing a worse classification performance. For this reason, when compound files are included in the preparation of a training set, we suggest to consider compound files but

ensuring to exclude any embedded data, as we do in the experiment presented below.

3.2 Experimental Setup

We proceed to compute the classifiers according to the methodology presented in [4], except for some key differences in the data set preparation, because as noted above we specifically take into consideration the compound file problems. We collected `bmp`, `doc`, `exe`, `gif`, `jpg`, `mp3`, `odt` and `pdf` files. For `bmp`, `gif`, `jpg` and `mp3`, we used the very same dataset of randomly downloaded files we used in [4]. Differently, for `doc`, `exe`, `odt` and `pdf`, we consider the embedded data issues. `doc`, `odt` and `pdf` are common document files with the possibility to embed multimedia data, but images and mouse cursors may also be found embedded in executable files. For such file types, we have modified the dataset used in [4] by discarding all those files containing embedded data, and by adding new files (without embedded data) randomly downloaded from the Internet with the same approach originally used to create the dataset. The files have been inspected to detect the possible presence of embedded data: document files have been manually inspected, and in the case of `pdf` files the `pdfimages` Linux tool has been also used. For executables, the ResourcesExtract tool [17] has been used. We simply excluded `ppt` files from our test, as it is basically impossible to find presentations without embedded data (e.g., pictures are almost always present).

The collected files are decomposed in 512 byte blocks, getting roughly 28000 blocks per each file type. Note that we select a 512 bytes length since it is the smallest block size commonly used to manage storage media, and we showed in [4] that smaller block size makes the classification task harder, so our results can be considered very conservative.

Using a framework built in Matlab, for each file type we compute a SVM classifier able to detect its blocks. For each classifier, it is necessary to set the relative values for parameters γ and C as explained in [4], as well as to select the features for the block representation. For the identification of the best feature/parameter combination for each file type we proceed as follows. For each file type we start from the collection of blocks we have obtained, and prepare 7 datasets split in a training and a test set, respectively containing 2000 and 8000 blocks. In each training set, half of the blocks are of the target file type, while the remaining blocks uniformly represent the other file types. In test sets, all file types are uniformly represented. For each file type we train and test a series of classifiers varying $\gamma = \{2^{-15}, 2^{-13}, 2^{-11}, \ldots, 2^{5}, 2^{7}\}$, $C = \{2^{-5}, 2^{-3}, 2^{-1}, \ldots, 2^{13}, 2^{15}\}$ and trying combinations of all of the features

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

Table 1. Feature set used to represent file blocks in the classification process.

mentioned at the beginning of this Section, as well some reduced versions (e.g., BFD related only to ASCII byte values) and concatenations (e.g., Entropy-Complexity-BFD). We refer the reader to [4] for the details. We test all of the possible feature/parameter combinations on each of the 7 datasets, and select the best combination using as a target function to maximize $0.5 \cdot TP + 0.5 \cdot (1 - FP)$, with TP and FP respectively "true positive" and "false positive" rates. Once the best combinations for each file type are identified, they are used to compute a final set of classifiers relying on the full block collection, with training sets of 28000 blocks and test sets of 112000 blocks.

3.3 Experimental Results

With regard to the first step, for the classifier feature/parameter exploration, the best set of features resulting for our block representation is summarized in Table 1. In particular their specific combinations, along with the associated SVM parameters, per each file type are presented in Table 2, where also the final classification results are shown. The concatenation Entropy-Complexity-BFD appears to be an effective representation for almost all the file types, except for `bmp`, for which RoC is marginally better. So, confirming the results of [4], file block representation can actually be the same for most different file types.

Classification results are quite good, and as expected they show an improvement compared to [4]. In fact, the average TP rate increased by 4%, but in particular the average FP rate is halved (from 12% to 6%). This is evidently the result of proper dataset preparation, and in particular of avoiding blocks from compound files with embedded data. This is particularly evident by observing the `doc` classifier FP rate, which decreases from 19.8% in the reference work to the current 2.4%, and in particular its specific FP rates against `gif` and `jpg` (two common types of embedded data in `doc` files) have been respectively pulled down by 6% and 28%. Symmetrically and for the same reason, the new dataset preparation improved also `gif` and `jpg` classifiers, with their specific

	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	-

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

FP rates against `doc` files decreased respectively by 3% and 15%. As a further example, during file collection we found and discarded several `exe` files embedding `bmp` images, and this contributed in reducing by 4% the number of `exe` blocks erroneously identified as `bmp` and by 10% the misclassification of `bmp` blocks as `exe`. So a first key finding is that the avoiding compound files embedding data in the preparation of file block training datasets appears to dramatically improve block classification precision when supervised learning approaches are adopted.

The classifiers, as they are, can support data recovery of primitive file types (e.g, `bmp`, `jpg`), since they can discern a large portion of their blocks; then the identified blocks could be concatenated in several ways to attempt file reconstruction. Differently, the recovery of compound files (e.g., `doc`, `pdf`) will require to deal with embedded data at some points, so the classifiers may provide only a limited support for their recovery. For instance, they could be useful to identify blocks not embedding any data, and could be used concurrently with primitive file type classifiers which would instead detect the embedded sections. They could also support the recovery of those compound files not containing any external data (e.g., a fully text-based doc). Anyway, for such file types, in the end

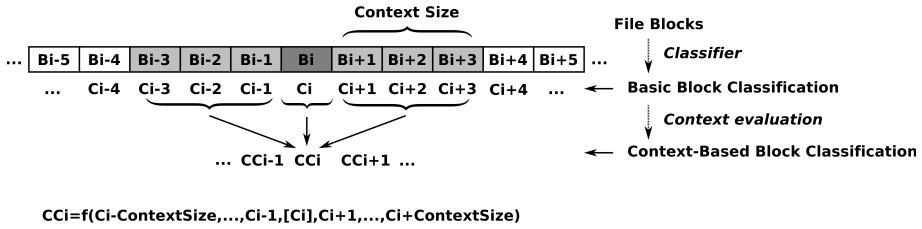

Figure 1. Context-based file block classification.

it will still be advisable to consider internal file structures as suggested in [16].

4. Context-Based Block Classification

Most of the files of interest in forensic recovery (e.g., documents, images, videos) are typically not small in size, and thus span multiple blocks (some statistics are presented in [2]). Usually, modern file systems tend to reduce fragmentation of files, meaning that blocks belonging to a same file are stored in contiguous locations as much as possible, while in case of fragmentation the most common scenario is bi-fragmentation, where a file is stored as two series of contiguous blocks far each other on the media [2]. 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 of its fragments. This is, fundamentally, the reason why basic file carving works most of time.

Block classifiers are typically not perfect (this is evident in all the literature reviewed in Section 1.2), and even in case of low error rates (as it happens for our classifiers working on primitive files), when handling terabytes of data, misclassification may have a considerable impact on final results. So an improvement in block classification precision is needed. Moreover, when there is not enough information for a correct decision, it is preferable to have a *not-classified* information instead of a wrong classification [16], to avoid inducing errors in those processes that exploit the classification information.

Our idea is to exploit file block contiguity: for the classification of a block B_i we rely on the classifications of its neighbouring blocks, because they tend to belong to the same file. We call such blocks the *Context* of B_i (Fig. 1). We set a *ContextSize* representing the number of neighbour blocks to consider on each side of B_i, and then we combine the classifications of the those blocks, along with the classification C_i of the block itself, to compute a *Context-Based* classification $CC_i = f(C_{i-ContextSize}, ..., C_{i-1}, [C_i], C_{i+1}, ..., C_{i+ContextSize})$. The idea

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
```

is that if a good, but not perfect, classifier is available, relying on a series of classifications (context) a more robust evaluation may be achieved.

4.1 Block Context Evaluation

Our classification model is binary: for a given target file type, for each generic block B_i we obtain a classification $C_i \in \{-1, 1\}$, where 1 means the block is of the target type, and -1 means vice-versa.

Our Context-Based classification CC_i for a block B_i is achieved according to Algorithm 1. The classifications of the context blocks preceding B_i (i.e. the "left" context) are processed through a *ContextEvaluation* function, whose returned value over the interval $[-1, 1]$ expresses how many of the considered blocks are target or non-target, respec-

tively with a positive or negative value. The context blocks following B_i ("right" context) are evaluated in the same way.

We propose two variants of the algorithm. In the first variant, the classification C_i of B_i itself is ignored, relying only on the context. In the second variant, we take into account C_i. In the first variant, the final classification CC_i is "target" if both the evaluations of the context express a positive value (this is interpreted as B_i being in the middle of a target file). Vice-versa, if the two evaluations present a negative value, the block is considered non-target. In the second variant, where C_i is taken into account, if its value agrees with the two context evaluations, the final classification is the same as in the first variant; otherwise, the output is "not classified", given the mismatch in the available information.

Some particular cases are taken into account. If one of the two context evaluations returns 0 (e.g., maybe because the context is placed over the blocks of two different contiguous files), the block classification C_i is compared with the non-null evaluation and if they agree the block is classified accordingly, otherwise not-classified is returned. Finally, if the two evaluations disagree or are both equal to 0 the block is labeled as not-classified, since a non consistent information is available on it.

We point out that, for the first block of a generic disk image, since the left context is null, $LeftEvaluation = 0$ by default; then for the following blocks the left context grows in size, up to the defined ContextSize, and it is considered for the final classification. Obviously there is a symmetric behavior at the end of the disk image, with $RightEvaluation = 0$ for the last block.

The *ContextEvaluation* functions that we test in our experiments are presented in Table 3. Using the *Uniform* function, we compute the average of the classifications of all of the blocks in a given context, without weights. The returned value directly expresses whether most of the context blocks are target or not. For the *Exponential* function, and even more so for the *Linear* one, the weights for the classifications of the blocks closer to B_i are higher: the idea is that such blocks are more likely to be part of the same file as B_i, and so their support for the final decision is increased.

We remark that, although in the next sections we experiment this solution relying on the SVM classifier we developed and trained in Section 1.3.2, the proposed approach is general, and can be applied to any content-based file block classifier, regardless of the specific underlying algorithm.

	$ContextEvaluation(C_1, C_2, ..., C_n)$
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 $weight = \sum_{i=1}^{n} 1 - \frac{1}{n}(i-\frac{1}{2})$

Table 3. Context evaluation functions.

4.2 Experimental Setup

We focus on improving the detection of `gif` files through the Context-Based classification. We choose them since they are a primitive file type whose classifier, as shown in Table 2, performs well. We want to show that a good classifier can be used as base in our context-based algorithm to achieve an enhanced classification.

For this experiment we collect a new data set, different from the one used for training and testing our classifiers. For each considered file type presented in Section 1.3.2 we downloaded from the Internet 1.5 MB of files, each roughly 100 KB in size. All such collected files are considered as 512 byte block sequences, and are used to compose 4 different "disk image" scenarios:

1 the files are randomly concatenated together forming a disk image "without fragmentation";

2 all files are split in two equal fragments that are then randomly concatenated together achieving a bi-fragmented disk scenario;

3 as above, but the fragmentation ratio is increased by splitting the files into 3 equal fragments;

4 as above, but the fragmentation is further increased by splitting the files into 10 equal fragments.

The idea is to test the context-based solution on common situations, represented by the first three cases [2], providing also a test in a possible, albeit unlikely, high fragmentation scenario, provided by the last case.

The experiment is made up of two stages: first, we use our `gif` classifier on all blocks of a generated disk image getting their basic classification and then, relying on such results, the Context-Based classification for each of them is computed. We use both the variants of Algorithm 1 as described in Section 1.4.1 (i.e. both considering and ignoring the classification C_i of the block itself), and we explore different Context-Size parameters (3, 5, 8 and 10). We do this for all of the 4 scenarios above.

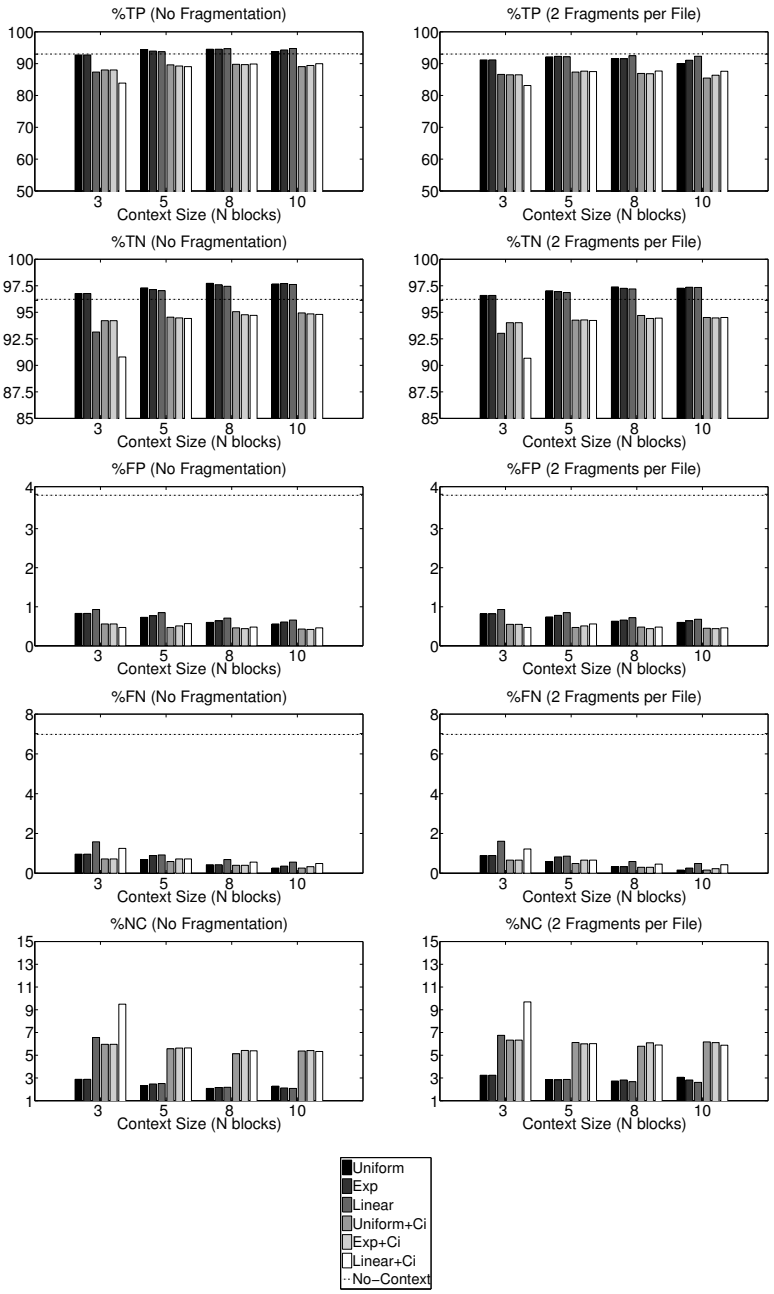

Figure 2. Context-based file block classification: no fragmentation and bi-fragmented file scenarios.

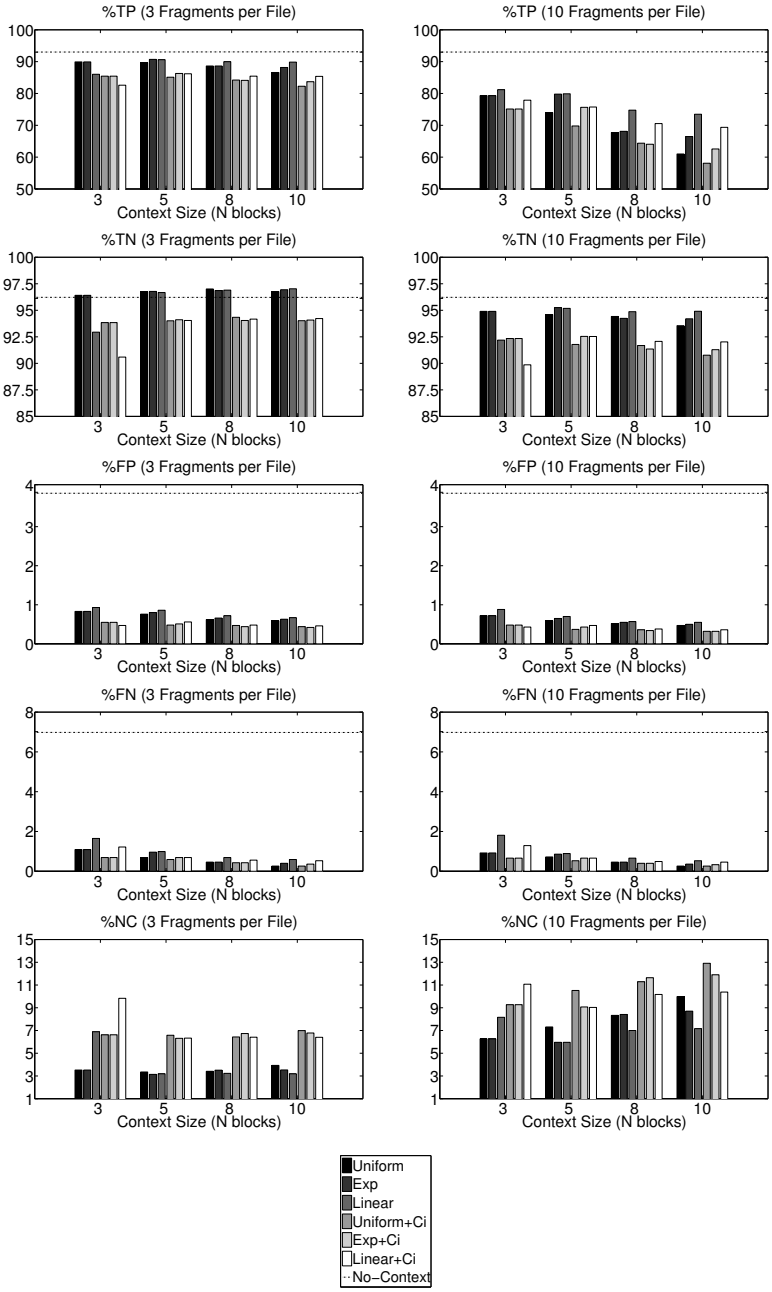

Figure 3. Context-based file block classification: 3 fragment-split file and 10 fragment-split file scenarios.

4.3 Experimental Results

The results of our experiments are presented in Figure 2 and Figure 3. Differently from the standard classification results (Table 2), for the Context-Based classification it is necessary to show both true negatives and false negatives rates (TN and FN respectively, in the following), along with the not-classified (NC) rates, since $TP + FN < 100\%$ and $TN + FP < 100\%$, due to the presence of NC blocks. We use the graphs to show also the basic classification results, achieved with the non-contextual `gif` classifier ("No-Context"), as a baseline.

The first evident and significant result concerns FP and FN rates: both are strongly improved (i.e., decreased) compared to the basic classification, in any given fragmentation scenario or parameter configuration. For the first three scenarios, in the variant ignoring the classification of the block itself, also the TN rates are improved, whereas the TP rate decreases for growing fragmentation rates, but it is increased on scenario 1 (and we remind the reader that non-fragmentation is the most likely condition for a file). Such results can be explained by the presence of long series of contiguous non-`gif` blocks representing 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 for target block identification. At the same time some non-classified blocks are present (roughly 3% of all blocks for the first three scenarios), mainly due to `gif` blocks not correctly classified as target; this is more evident looking at the graphs of the last fragmentation case, where some losses in the TP rate are correlated with the increased NC rate. In other words, with our proposed scheme we are reducing the classification errors (a desired result), at the expense of increasing the number of non-classified blocks.

As expected, classification performance gets worse increasing the fragmentation rate, but we observe that thanks to our methodology TP and TN rates tend to be converted to NC, rather than to FP and FN, keeping such rates low, therefore preserving a reduced classification error.

Increasing the ContextSize helps to reduce FP and FN rates, 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 $ContextSize = 5$ appears to represent a good trade-off in our experiments.

Our results suggest that it is better to use the variant which ignores the classification of the block itself. In fact, taking C_i into account

only affects negatively TP and TN, leaving FP and FN substantially similar, due to an increased NC rate.

With regard to the different proposed *ContextEvaluation* functions we observe that they appear to perform comparably well. The Linear one is worse than the others for $ContextSize = 3$, probably because it does not exploit effectively the short context focusing mainly on the two contiguous blocks of B_i. On the other hand, the Linear function outperforms the others in case of high fragmentation; in such cases, it seems effective in reducing the effect of non-target blocks in the context, a typical situation in proximity of fragment boundaries, relying on the context blocks closer to the block currently under classification.

With the proposed classification approach NC blocks will be concentrated mostly at the boundaries of files/fragments. As a potential application, a modern file carver can also exploit this classification scheme to identify block regions belonging to a specific file type, related to a file/fragment on the storage media, but with "faded" boundaries: correct classification in the middle of fragments with NC values in proximity of their ends. Then the carving system could attempt to collate together such block regions, varying their length in the range identified by the relative NC areas, until a complete file is recovered (the different combinations may be for instance checked out by using file validators [2]).

5. Conclusions and Future Work

In this paper we proposed and validated two approaches to overcome two challenging problems typically associated with file block classification in forensic data carving, namely the existence of compound file types and the errors induced by false positives and negatives in classification and in the following reconstruction attempt.

We firstly presented comparative experiments that demonstrate the issue of compound files in file block classification. We showed how the composition of different file types impacts classification performance, and suggested to exploit only compound files without embedded data in classifier training set preparation.

Then, we introduced a methodology to generally improve the performance of file block classifiers. Our approach works by exploiting the spatial coherence of data, i.e. the contiguity of blocks related to the same file. We call this approach a "context-based" classification. We showed by experimentation that it improves block classification performance, in particular by keeping misclassification low, with a limited cost in terms of non-classified blocks.

While the present paper builds on a specific classifier as a working example, the approach we proposed is completely general, and can be easily applied to any content-based file block classification algorithm. In a future extension of this work, we will try to assess the impact of this approach on other block classification strategies.

Acknowledgments

This work has been partially supported by the "Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks" programme of the European Commission, under the project "i-Code: Real-time Malicious Code Identification". The research leading to these results has also received partial funding from the EU Seventh Framework Programme (FP7/2007-2013) under grant agreement nr. 257007 – "SysSec". This publication reflects the views only of the authors, and the EC cannot be held responsible for any use which may be made of the information contained therein.

References

[1] G.G. Richard III and V. Roussev, "Scalpel: a frugal, high performance file carver", *Proceedings of the 2005 Digital Forensics Research Workshop (DFRWS)*, 2005.

[2] S.L. Garfinkel, "Carving contiguous and fragmented files with fast object validation", *Proceedings of the 2007 DFRWS Conference*, pp. 2–12, 2007.

[3] A. Pal and N. Memon , "The evolution of file carving", *IEEE Signal Processing Magazine*, vol. 26, no. 2, pp. 59–71, 2009.

[4] L. Sportiello and S. Zanero, "File block classification by Support Vector Machines", *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pp. 307–312, 2011.

[5] M. McDaniel and M. H. Heydari, "Content Based File Type Detection Algorithms", *Proceedings of the 36th IEEE Annual Hawaii International Conference on System Sciences*, 2003.

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

[7] S.J. Moody and R.F. Erbacher, "SADI - Statistical Analysis for Data Type Identification", *Third International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 41–54, 2008.

[8] M. Karresand and N. Shahmehri, "Oscar-file type identification of binary data in disk clusters and RAM pages", *Security and Privacy in Dynamic Environments*, pp. 413–424, 2006.

[9] M. Karresand and N. Shahmehri, "File type identification of data fragments by their binary structure", *IEEE Workshop on Information Assurance*, pp. 140–147, 2006.

[10] S. Axelsson, "Using Normalized Compression Distance for classifying file fragments", *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pp. 641–646, 2010.

[11] S. Axelsson, "The Normalised Compression Distance as a file fragment classifier", *Proceedings of the 2010 DFRWS Conference*, pp. S24–S31, 2010.

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

[13] W.C. Calhoun and D. Coles, "Predicting the types of file fragments", *Proceedings of the 2008 DFRWS Conference*, pp. S14–S20, 2008.

[14] C.J.C. Burges, "A tutorial on support vector machines for pattern recognition", *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[15] C.W. Hsu, C.C. Chang and C.J. Lin, "A practical guide to support vector classification", *Department of Computer Science and Information Engineering, National Taiwan University, Taiwan*, 2003.

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

[17] Nir Sofer, "ResourcesExtract v1.16", *http://www.nirsoft.net/*, 2011.