machine learning & knowledge extraction

MDPI

Article

Hierarchy-Based File Fragment Classification

Manish Bhatt [1,†], Avdesh Mishra [2,†], Md Wasi Ul Kabir [1,†], S. E. Blake-Gatto [1], Rishav Rajendra [1], Md Tamjidul Hoque [1,*] and Irfan Ahmed [3]

[1] Department of Computer Science, University of New Orleans, 2000 Lakeshore Dr., New Orleans, LA 70148, USA; mbhatt@my.uno.edu (M.B.); mkabir3@my.uno.edu (M.W.U.K.); eblanken@uno.edu (S.E.B.-G.); rrajendr@my.uno.edu (R.R.)

[2] Department of Electrical Engineering and Computer Science, Texas A&M University-Kingsville, Kingsville, TX 78363, USA; Avdesh.Mishra@tamuk.edu

[3] Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284, USA; iahmed3@vcu.edu

* Correspondence: thoque@uno.edu

† Equal Authorship.

Received: 22 June 2020; Accepted: 30 July 2020; Published: 3 August 2020

Abstract: File fragment classification is an essential problem in digital forensics. Although several attempts had been made to solve this challenging problem, a general solution has not been found. In this work, we propose a hierarchical machine-learning-based approach with optimized support vector machines (SVM) as the base classifiers for file fragment classification. This approach consists of more general classifiers at the top level and more specialized fine-grain classifiers at the lower levels of the hierarchy. We also propose a primitive taxonomy for file types that can be used to perform hierarchical classification. We evaluate our model with a dataset of 14 file types, with 1000 fragments measuring 512 bytes from each file type derived from a subset of the publicly available Digital Corpora, the govdocs1 corpus. Our experiment shows comparable results to the present literature, with an average accuracy of 67.78% and an F1-measure of 65% using 10-fold cross-validation. We then improve on the hierarchy and find better results, with an increase in the F1-measure of 1%. Finally, we make our assessment and observations, then conclude the paper by discussing the scope of future research.

Keywords: file fragment classification; support vector machines; hierarchical classification; digital forensics; file carving

1. Introduction

It is essential for a forensic investigator to be able look at an artifact, which can be a network packet or a piece of data, and readily recognize what kind of data it is. This capability is utilized by investigators to analyze break-ins, making sense of the residual data and attacker footprints, decoding memory dumps, reverse engineering malware, or merely to try to recover crashed data from a drive. As investigations increase in complexity, the need for a tool that automatically classifies a file fragment becomes crucial. It is common knowledge that files are not stored in contiguous space on the disk but are often stored as non-contiguous disk fragments. Moreover, the information is stored as files in two areas in computer disks—allocated and unallocated space. Unallocated space is used to store new data, which may contain deleted documents, file system information, and other electronic artifacts. The unallocated space on the disk is important from an investigation perspective because it contains significant information in a deleted form [1]. This information can be recovered and analyzed by a forensic investigation. Often the recovered file is just a fragment of the original file. File fragment classification techniques can be effectively used to classify the file fragments.

The file fragment classification problem refers to the problem of taking a file fragment and automatically detecting the file type. This is an important problem in digital forensics, particularly for carving digital files from disks. The problem with file fragment classification is that it is quite complicated due to the sheer size of the search space [2]. Moreover, the definition of the file type can be quite vague, and often file types are only characterized by their header information.

In the past, several statistical [3], rule-based [4], or machine learning approaches [5–12] have been proposed as solutions for the file classification problem. In the machine learning description of the problem, each file type is thought to be a category (class) and certain features that are thought to characterize the file fragment are extracted. Then, supervised machine learning approaches are used to predict the category label for each test instance. Some of the methods also incorporate unsupervised machine learning approaches. For example, Li et al. [13] applied the k-means clustering algorithm to generate models for each file type and achieved good classification accuracy. In previous work, the histogram-based approaches, the longest common subsequence-based approaches [14], and natural language processing-based approaches [15] have constituted the majority of the work done in this realm. Furthermore, other approaches based on the transformation of the fragment space into images have also been proposed [16].

Regarding the machine learning approaches, several works have included individual classifiers, ranging from naive Bayes to neural networks. As opposed to having a single classifier, the advantages of having a hierarchical classifier are outlined in several areas [17–19].

In this work, we propose a classification technique called hierarchical classification to classify file fragments without the help of file signatures present in headers and footers. Since there are many different kinds of files, it has also been suggested that a generic approach towards file fragment classification will not provide desirable results, and more specialized approaches for each file type are required [2]. As opposed to having a single multi-class classifier, such as decision trees, a hierarchical classifier maps an input feature set to a set of subsumptive categories. Due to the no-free-lunch theorem [20], there cannot be a single model that can be used for all categories. Other ensemble learning algorithms such as random forests could also be used in this problem domain; however, they do not directly provide coarse-to-fine-grained classification. In our approach, at every hierarchy level and at every node, a hierarchical classifier creates a more specialized classifier that performs a more fine-grained classification task than the previous node. In a way, the classification process at each node is more specialized than the classification process at each preceding node. The classification process first occurs at a low level with particular pieces of input data. The classifications of the individual pieces of data are then combined systematically and classified at a higher level iteratively until a single output is produced, which is the overall classification of the input data. Depending on application-specific details, this output can be one of a set of pre-defined outputs, one of a set of online learned outputs, or even a new novel classification that has not been seen before. Generally, such systems rely on relatively simple individual units of the hierarchy that have only one universal function to do the classification. Thus, hierarchical systems are relatively simple, easily expandable, and are quite powerful.

In this work, we use the hierarchical classification technique for 14 different file types by taking support vector machines (SVM) [21] as our base classifiers to classify file fragments. During the experimentation procedure, we use the Garfinkel corpus as our test and train dataset [22] and extract certain standard features [15] from it. Moreover, we evaluate our hierarchical classifier based on SVM against 512-byte-sized fragments from the corpus, showing that our classifier has the highest accuracy of 68.57% and an F1-measure [23] of 65%. The hierarchical classifier outperformed many of the other proposed classifiers, most significantly the one proposed by Xu [16] by almost 20%. After hierarchy refinement, we were able to increase the F1-measure to 66%.

Hence, the contributions in this paper are two-fold. First, we suggest a hierarchical taxonomy of files that can be used for hierarchical classification. Secondly, we propose a local classifier per node

approach by using SVM as our base classifier. We find that this approach, although unrefined, opens up a different way of looking at the file fragment classification problem.

This paper is organized as follows. First, we introduce the plethora of related work that has been performed to solve the file fragment classification problem. Then, we provide a theoretical background regarding SVM and our hierarchical classification approach. After this, we talk about the configuration details of our experiments and present our evaluation results. Finally, we compare our results with the existing techniques which have been proposed in the literature, conclude the paper, and describe future works.

2. Background Work

This section presents a review of directly comparable previous works on the file fragment classification problem and hierarchical classification technique.

2.1. File Fragment Classification

Ever since the dawn of digital forensics, the file fragment classification problem has been studied quite extensively. Probably the most often used file identification command in history is the Unix file command and the libmagic library [4]. The file command works by comparing specific regions of a file (most often the file's header and footer) against a database and reports the first match. This command is reasonably accurate when dealing with complete files. However, it indicates that a file fragment is a "data" file when it is provided with a file fragment.

The first non-header- and non-footer-based file identification techniques appear to have been proposed by McDaniel in 2003 [24], taking into account whole files. His approach consisted of the creation of a histogram of frequencies of American Standard Code for Information Interchange (ASCII) code for each file, while his corpus consisted of a total of 120 files from 30 different file types. The histogram was then used as a feature vector and a clustering algorithm was applied. His approach yielded a 27.5% true positive (TP) rate for the byte frequency analysis (BFA) algorithm and 46% for the byte frequency cross-relation (BFC) algorithm [24]. Li modified this algorithm, which is essentially 1-g analysis, to formulate a centroid-based approach derived from byte frequency distribution as the signature of the file type. Interestingly, this approach yielded a near-perfect accuracy for 20-byte fragments, but the full file type accuracy dropped significantly [13]. Karresand and Shahmehri developed a similar method, which provided greater consistency regarding accuracy [25].

With regards to more specialized approaches as opposed to merely looking for a set of signatures, researchers have used a combination of machine learning techniques and statistics. Researchers typically assemble a corpus of files that constitute various files downloaded from the Internet, images of hard drives that are no longer used, or use standardized corpora like the Garfinkel corpus [22]. Such corpora are divided into a train set and a test set. The file fragments in the train set are pre-processed and adequate features are extracted via statistical techniques and fed into traditional machine learning algorithms [2]. The algorithm is then able to take the test dataset as the input and categorize it into one of several classes.

Conti et al. used a private dataset of 14,000 512-byte fragments, which they characterized using statistical measurements such as Shannon's Entropy, the Hamming weight, and Chi-square goodness of fit. They classified each of these vectors against the remaining 13,999 vectors using the k-nearest neighbor (kNN) approach and used Euclidian distance as the distance measure. They achieved significantly higher accuracy than their predecessors, however their method did not work as expected with real-world data [6].

Axelsson [5] classified 28 different types of files using the kNN algorithm with the nearest compression distance as the distance metric. File fragments were generated from the Garfinkel corpus [22]. His experiments consisted of 10 trials, wherein for each trial 10 files were selected and 512-byte fragments were extracted at random. The fragments were then classified against a dataset of 3000 fragments with known types. The average classification accuracy in this case was around 34%.

Calhoun and Coles considered only four types of files (jpg, bmp, gif, and pdf) in their work, in which linear discriminant analysis was used for every pair of file types [14]. The features considered were various statistical measurements, such as Shannon's entropy [26] and the frequency of the ASCII codes. They achieved a reasonably high accuracy of 88.3%. However, since they classified fragments based on only four types of files on a pairwise basis, we are not sure if this technique could be generalized because they considered only a limited number of cases. Moreover, other significant tests and assessments of this technique on file fragments of different types would be required to access the validity of this technique. We also cannot compare it directly with our approach.

Xu et al. [16] proposed the transformation of a fragment into a grayscale image to derive the GIST feature (which is a descriptor, which representation of a low-dimensional image that contains enough information to identify the scene in an image) set from the image, using a classifier to classify the fragments. They achieved an accuracy of 54.7% via a type-unbiased model.

Veenman used unigram counts, Shannon's entropy, and the algorithmic complexity as features for linear discriminant analysis to classify fragments, which were 4096 bytes in size. Their average classification accuracy was reported to be 45% and their private dataset consisted of 11 different file types and 3000–20,000 fragments per file [27]. Fitzgerald used similar features used by Conti along with bigram counts and fed his feature set into SVMs to achieve an average prediction accuracy of 47.5%. Using the macro-averaged F_1 metric [28], they averaged 46.3%, with a standard deviation of 1.65% over 10 runs [15].

Wang et al. [29] proposed an approach using sparse coding for automatic feature extraction. They created a sparse dictionary for higher-order N-grams (N = 4, 8, 16, 32, 64). These dictionaries were then used to estimate N-gram frequencies for a given file fragment. They achieved an average accuracy of 61.31% for 18 file types using support vector machines (SVMs).

Several researchers also applied deep learning to solve the file fragment classification problem. Chen et al. [30] proposed a file fragment classification technique using fragment-to-grayscale image conversion and deep learning. They converted 512-byte fragments into 64×64 images and then used the convolutional neural networks (CNN) to create the model. The model was trained and tested on the 16 file types form the public dataset GovDocs and was reported to achieve 70.9% accuracy. Mittal et al. [31] also relied on a convolutional neural network for large-scale file fragment classification. They created a new dataset with 75 file types. Their method achieved an average accuracy of 77.5% on the new dataset.

2.2. Hierarchical Classification

Over time, several researchers have worked in the realm of hierarchical classification. Stojanova et al. [32] proposed a global-based method to consider self-correlation, i.e., the statistical relationships between the same variable at different but related instances. During training, a combination of features and self-correlation among cases is used. The network models self-correlations, which are used by the method while learning.

Borges et al. [33] proposed a global-based competitive neural network formed by an input layer and an output layer. The neural network weights are adjusted according to the classes associated with the winner neurons.

Sun formulated the classification task as a path selection problem, where each path starts on the root and terminates on a leaf or internal node. They used partial least squares to solve the label prediction problem as an optimal path prediction problem [18].

Beebe et al. [34] proposed a two-level hierarchical classification model. The file fragment first classifies data and file types into classes, and then it classifies them into class-based types. They applied exploratory data analysis using the k-means and expectation maximization clustering algorithms to derive alternative hierarchical models from the data. The proposed method achieved good classification accuracy with six classes and 52 file types due to having a simple two-level hierarchy.

According to the literature [18,32–34], hierarchical classification based on machine learning is a promising new technique that can be used to solve the file fragment classification problem.

3. Theoretical Background

In this work, we utilize a hierarchical classification technique, using an optimized SVM as the base classifier to classify file fragments.

3.1. Hierarchical Classification

In traditional classification, a model is trained to assign a single class to an instance. When it comes to hierarchical classification (HC), the classes are structured in a hierarchy containing subclasses and superclasses. Depending on how complex the classification problem is, the hierarchical taxonomy can be a directed acyclic graph (DAG) or a tree, and an instance can be classified into one or more paths of this hierarchy.

Considering X as the space of instances, an HC problem consists of learning a function (classifier) f able to classify an instance $x_i \in X$ into a set of classes $C_i \in C$, with C being the set of all classes in the problem. The function f must respect the constraints of the hierarchical taxonomy and optimize an accuracy criterion. The limitations on the hierarchy hold such that if the class is predicted, its superclasses get predicted automatically.

Two main approaches have been used to deal with HC problems—local and global. In the local approach, traditional algorithms such as SVM or neural networks are used to train a hierarchy of classifiers. These are then used to classify instances following a top-down strategy. In the top-down strategy, a classifier is associated with a class node and is responsible for distinguishing between the child classes of this node. A test instance is passed through all classifiers, from the root until the deepest predicted class, which can be a leaf-node (mandatory leaf-node classification) or an internal node (non-mandatory leaf-node classification). Different strategies can be used in the local approach: one local classifier per node (LCN), one local classifier per parent node (LCPN), and one local classifier per level (LCL). While LCN uses one binary classifier for each class, LCPN associates a multi-class classifier to each parent node, which is used to distinguish between its subclasses. In the LCL strategy, one classifier is trained for each hierarchical level, predicting the classes of its associated level [19]. The global approach, in contrast, trains a unique classifier for all hierarchical classes. New instances are then classified in just one step.

Both local and global approaches have advantages and disadvantages. The local approach is easier to implement and more intuitive, since discriminating classes level-by-level resembles the process a human would do. Furthermore, the power of any classifier can be used and different classifiers can be combined. A disadvantage of this approach is that in the top-down strategy, errors at higher levels of the hierarchy are propagated to lower levels as the hierarchy is traversed towards the leaves. The global approach, in turn, avoids this error propagation and is usually computationally faster than the local approach. Additionally, it generates less complicated models than the combination of many models produced by the local approach. However, it is more complex to implement and does not use local information that may be useful to explore different patterns in different hierarchical levels. Because the information used to classify an instance in higher levels is different from information used to classify the same instance into deeper levels, the use of local information may improve the classification [35–37]. In our work, we use an optimized SVM classifier as our local classifier per parent node.

3.2. Hierarchy Definition

The first problem to tackle is the definition of the hierarchy. In our case, we have 14 different file types from which we extracted the fragments. We define a primitive hierarchy for the file types, as shown in Figure 1. The main idea behind defining a hierarchy is the following. In the first level, the files are differentiated according to their complexity, i.e., complex files are grouped as the first child and the non-complex files are grouped as the second child. Complex files are the files that allow

other simple files to be embedded in them, such as pdf and doc files. If the files are non-complex, then they are further classified into text files vs. binary files. This classification is valid, as the files are either text or binary. Moreover, the binary files can be further subclassified into compressed files and non-compressed files. If we have a new file that we need to assign to this hierarchy, we can follow the above-specified guidelines.

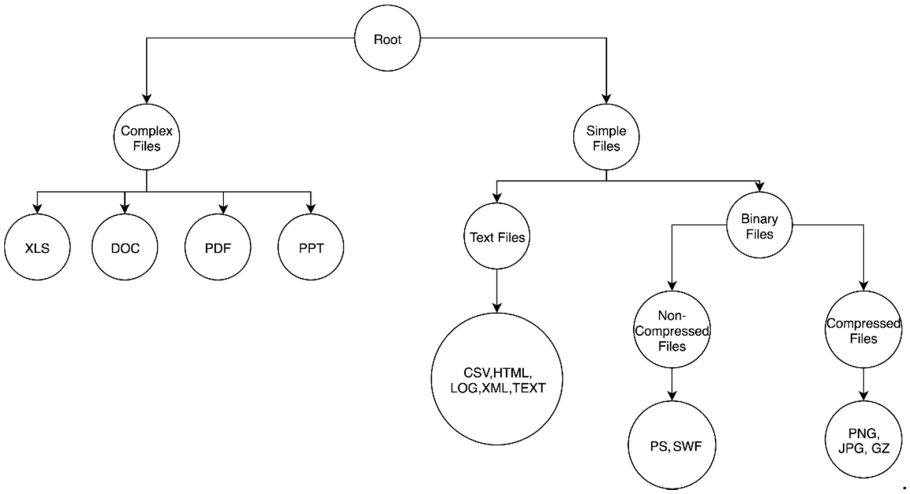

Figure 1. Hierarchy illustration for the files.

For the second hierarchy shown in Figure 2, we differentiated the text file branch by considering if the file was structured or unstructured, i.e., if they contained tags (e.g., HTML/XML tags) that structured different parts of the files or not. Furthermore, binary files were specified as being high entropy files or low entropy files. High or low entropy files are the files that have high unigram entropy or low unigram entropy, respectively. Under high entropy files we differentiated between archives and images, while under low-entropy files we distinguished between images and other file-encoded formats.

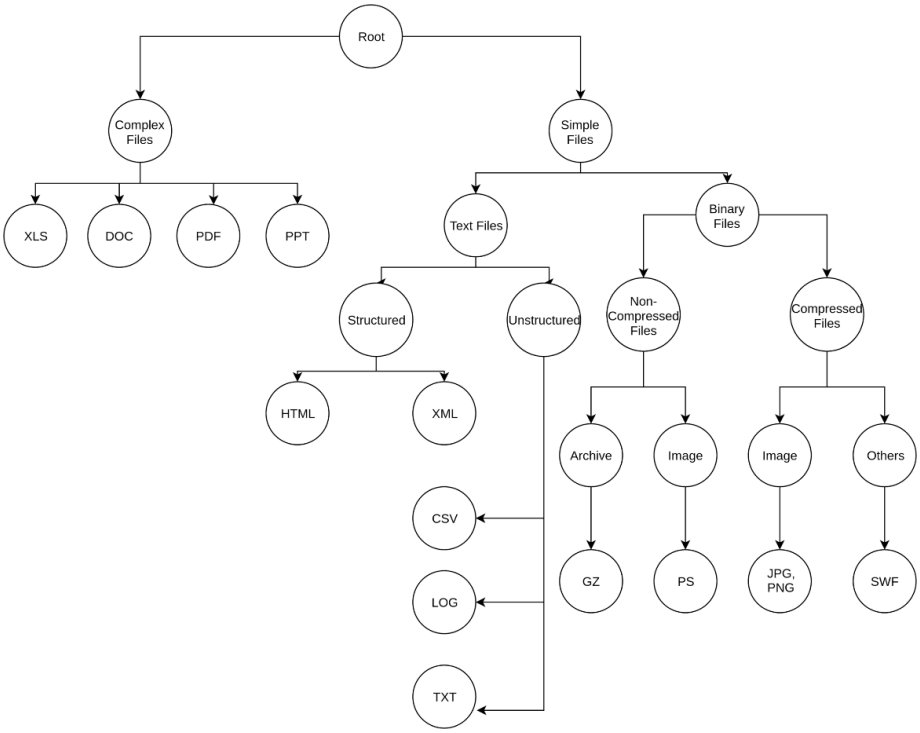

Figure 2. Hierarchy illustration for the files for experiment 2.

A critical advantage of defining a hierarchical classification such as this is that if an investigator takes one file fragment and does one pass through the classifier, they can categorically understand the content of the file. For instance, if a gz (GNU Zip file type) fragment has been classified as a jpg fragment, and the path followed is root:binaryfiles:highentropy:image:jpg, the investigator can know that the file fragment is high entropy even, though the file fragment was misclassified at the final level. This enables the user of such a classifier to obtain partial information about the file fragment correctly, even if the final information might be incorrect.

Finally, encrypted files are beyond the scope of our present work because the features we used are not characteristic of encryption. However, the proposed technique could be used to identify encrypted file types after re-engineering features.

3.3. Feature Descriptions

We extracted a total of 10 features from each fragment, as shown in Table 1. We used a feature ranking metric called information gain and found that these features were the most relevant, with metric values ranging from 0.229 ± 0.003 to 3.807 ± 0, and with the lowest being the Hamming weight and the highest being for the mean of the unigram counts. Hence, all the extracted features are relevant to the problem at hand.

Table 1. Information gain for feature selection.

No.	Feature	Information Gain Metric
1	Unigram Mean	3.807 ± 0.000
2	Unigram Standard Deviation	0.595 ± 0.003
3	Contiguity	1.448 ± 0.003
4	Mean Byte Value	1.464 ± 0.005
5	Longest Streak	1.590 ± 0.006
6	Compressed Length	0.841 ± 0.003
7	Hamming Weight	1.280 ± 0.005
8	Entropy in Bigram Distribution	0.229 ± 0.003
9	Bigram Mean	1.294 ± 0.004
10	Bigram Standard Deviation	0.595 ± 0.003

3.4. Unigram Count Distribution

At the byte level, we only have 256 different values, regardless of the ASCII range. Unigram count distribution refers to the frequency of occurrence of these values in the file. Moreover, to decrease the number of dimensions and to escape the curse of dimensionality [38], instead of using all 256 values in our feature, we fit this set to a normal distribution. As a normal distribution can be characterized completely by its mean and standard deviation, we use these two values per fragment as our features.

3.5. Entropy and Bigram Distribution

Similar to the Unigram Count Distribution, in this case, we find the frequency of occurrence of two-byte sequence (bigram) in the file, fit it to a normal distribution, and use the mean and standard deviation after fitting as our feature vectors. Furthermore, we also compute Shannon's entropy [26] for the bigram distribution and use it as another dimension in our feature space.

3.6. Contiguity

We computed the average contiguity between bytes (i.e., the average distance between consecutive bytes) for each file fragment.

3.7. Mean Byte Value

The mean byte value was one of the features included by Conti et al. [6]. The mean byte value is simply the sum of the byte values in a given fragment divided by the fragment size.

3.8. Longest Streak

The largest number of continuous occurrences of the same byte in a file is known as its longest streak.

3.9. Compressed Length

We used the compression length as one of our features to approximate the Kolmogorov complexity of the file fragment [27].

3.10. Hamming Weight

We calculated the Hamming weight of a file fragment by dividing the total number of ones by the total number of bits in the file fragment.

3.11. Evaluation Metrics

To evaluate our classifier on the dataset, we defined the following metrics. The i in the equations below refers to the instances.

3.12. Precision

Precision is defined as the ratio of the total number of true positives to the sum of the true positives and false positives, as in the following equation:

$$Precision = \frac{\sum_i TP}{\sum_i TP + FP} \tag{1}$$

The recall is also known as the true positive rate and is defined as the ratio of the total number of true positives to the sum of the total number of true positives and false negatives.

$$Recall = \frac{\sum_i TP}{\sum_i TP + FN} \tag{2}$$

3.13. F1-Measure

The F1-measure provides an average metric to the classification metrics of precision and recall. The F1-measure is defined as the harmonic mean between precision and recall. It can be represented mathematically as:

$$F1\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

4. Experiment Details

We downloaded the zip files labeled *000.zip–073.zip* from the Garfinkel corpus [22]. Then, we dropped the first and last byte sets to ensure the headers and footers do not skew the results. We created 512-byte-sized fragments from each file type and created our dataset, which consisted of the file types and fragments, as shown in Table 2. Then, we randomly sampled 1000 pieces of fragments from each file type to achieve a balanced dataset, as directly using the unbalanced dataset in a model can cause temporary or permanent bias. We extracted the features mentioned in the previous section from the file fragments. Finally, we defined our hierarchy, as shown in Figure 1, and performed 10-fold cross-validation on our model with the dataset. Upon optimizing the SVM parameters using grid search, we found that we got the best results for the two parameters of the Radial Basis Function (RBF)

Kernel: cost *C = 75*, *gamma = 0.0001*, where our search space for C was [0, 100] and for gamma was [0.0001, 2].

Table 2. Illustration of the dataset.

No.	File Type	Number of File Fragments
1	csv	19,340
2	doc	109,735
3	html	11,036
4	pdf	246,277
5	ppt	254,241
6	xml	1847
7	xls	302,304
8	txt	14,474
9	gif	21,443
10	jpg	25,022
11	png	1950
12	ps	63,149
13	swf	1010
14	gz	14,000

After conducting the experiment on a primitive hierarchy, as shown in Figure 1, we decided to further extend our hierarchy on the binary branch and came up with another hierarchy (Figure 2). We tested our hierarchical classifier against the dataset using our modified hierarchy and obtained a second set of results.

We performed our experiments on the server made of 48 processors and 256 GB of RAM. We implement our proposed hierarchical classification algorithm in Python (Scikit-learn libraries) and ran the experiments on a Linux operating system. Table 3 shows the time requires for each step of the experiments. The most time-consuming part is the grid search, which is used to find the best parameter (C and Gamma) for the support vector machine (SVM) algorithm. This process takes around 8.5 days to complete. After selecting the best parameter, we trained and tested the dataset using 10-fold cross-validation, which takes only 183.12 s to complete. To improve the performance of our algorithm, we ran 10-fold cross-validation in parallel using all 48 processors in the server.

Table 3. The execution time of our proposed method.

Steps	Time (sec)
Dataset Preprocessing	303.27
Feature Extraction	1640.60
SVM Grid Search (C and Gamma)	750,067.71
Train and Test with 10-fold CV	183.12

As our dataset is not that large, we did not compare our method with a deep learning algorithm. Deep learning requires high computational resources [31] and a large amount of data to perform well. The computational complexity of the deep learning method depends not only on the dataset size but also on the network architecture. A deep learning network may consist of many hidden layers and nodes, which makes it computationally expensive to train. Often the models are trained using graphics processing units (GPUs) rather than CPUs to overcome this problem. It is possible to achieve better performance using non-deep-learning methods with much less computational resources and less data [39].

5. Evaluation Results

5.1. Tuning SVM Parameters

As described in the experimental details, we first tuned the local classifiers (i.e., the SVMs) to derive the best parameters for the cost and gamma.

5.2. Effects of the Cost Parameter and Gamma Parameter

The effects of both the cost (*C*) parameter and the gamma parameter on the F1-measure are shown in Figures 3 and 4, respectively. The data were automatically fitted to a polynomial trend line. We can see in Figure 3 that as the cost parameter increases, the value of the F1-measure increases. On the other hand, in Figure 4, as the value of the gamma parameter increases for a specific value of C, the value of the F1-measure decreases.

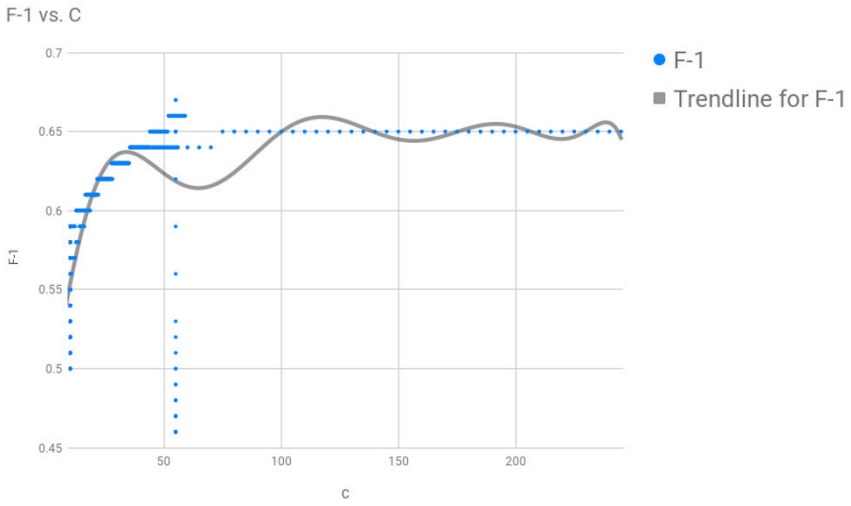

Figure 3. Graph of the cost parameter C vs. the F1-measure.

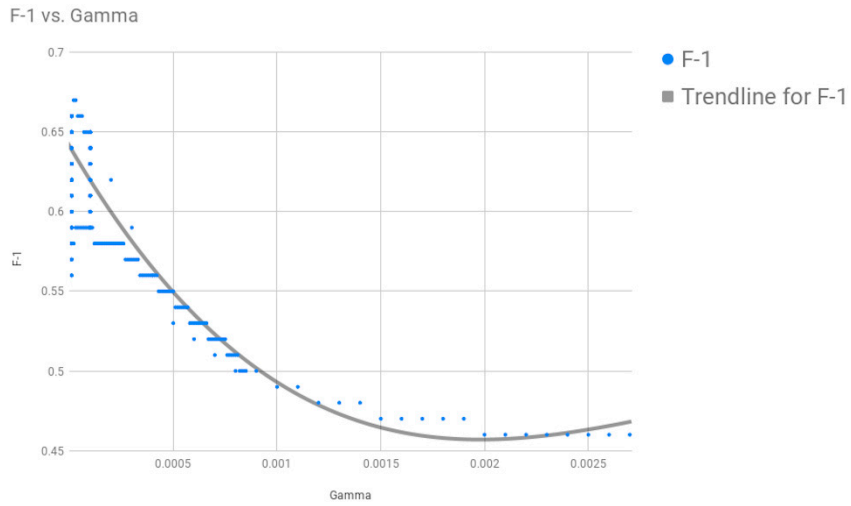

Figure 4. Graph of the gamma parameter vs. the F1-measure for cost C = 75.

5.3. Evaluation Metrics

After training the first hierarchical model on the 512 byte data, on average we were able to achieve a true positive rate (TPR) of 67%. Furthermore, our macro-averaged F1-measure was 65%, as evident in Table 4. Individually, the highest TPRs were seen for csv files, text files, png files, and swf files. Moreover, the highest F1-measures were observed for csv files, text files, and swf files.

Further experimental results of the classification using the first hierarchy (Figure 1) are provided in the confusion matrix in Figure 5. Finally, the worst TPR rates were seen for ppt, pdf, and doc files, which are grouped under the complex file category, as per our hierarchy.

Table 4. Classifier results for hierarchy 1.

File Type	Precision	Recall	F1-Measure
csv	0.89	0.91	0.90
doc	0.68	0.32	0.43
gif	0.81	0.83	0.82
gz	0.63	0.83	0.72
html	0.71	0.75	0.73
jpg	0.58	0.70	0.63
pdf	0.31	0.08	0.13
png	0.41	0.89	0.56
ppt	0.33	0.07	0.11
ps	0.52	0.73	0.61
swf	0.87	0.88	0.87
txt	0.86	0.87	0.86
xls	0.86	0.73	0.79
xml	0.86	0.87	0.87
Average Values	0.67	0.67	0.65

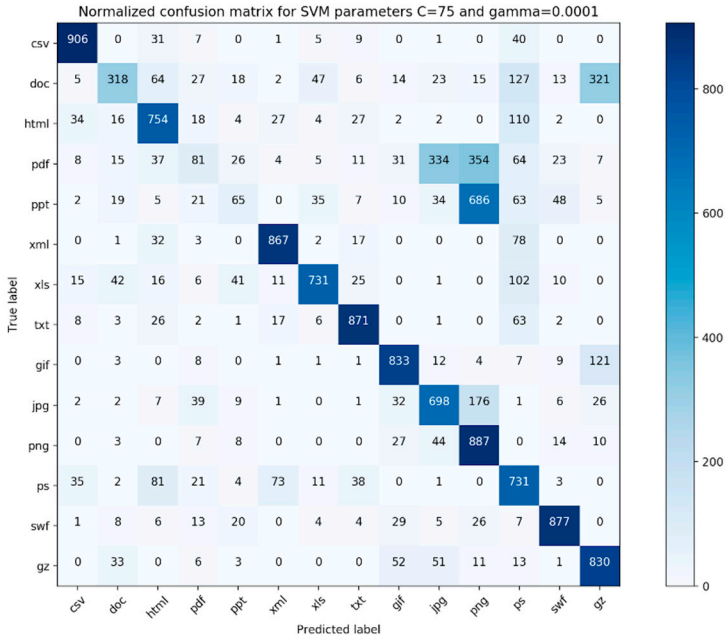

Figure 5. Confusion matrix for the classification of the files using hierarchy 1.

In the second set of results, we saw that by making the hierarchy more detailed, we were able to obtain slightly better individual accuracy for some of the complex file fragments, such as doc, pdf, and ppt. The improvement in classification is evident in the confusion matrix, as plotted in Figure 6. Table 5 also shows that the macro-averaged F1-measure was increased to 66%. However, no significant improvements were seen for simple file types.

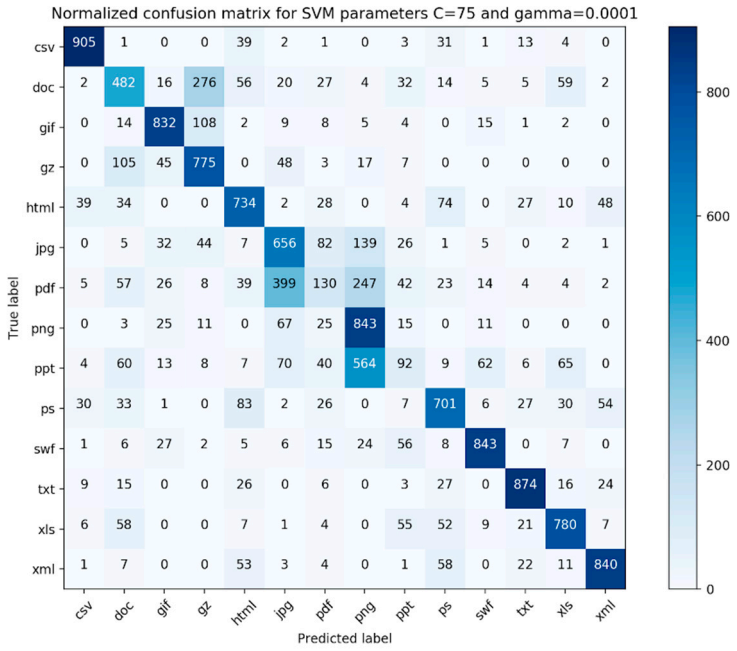

Figure 6. Confusion matrix for the classification of the files using hierarchy 2.

Table 5. Classifier results for hierarchy 2.

File Type	Precision	Recall	F1-Measure
csv	0.9	0.91	0.90
doc	0.55	0.48	0.51
gif	0.82	0.83	0.82
gz	0.63	0.78	0.69
html	0.69	0.73	0.71
jpg	0.51	0.66	0.57
pdf	0.33	0.13	0.19
png	0.46	0.84	0.59
ppt	0.27	0.09	0.14
ps	0.70	0.70	0.70
swf	0.87	0.84	0.86
txt	0.87	0.87	0.87
xls	0.79	0.78	0.78
xml	0.86	0.84	0.85
Average Values	0.66	0.68	0.66

6. Comparison with Previous Works and Discussion

The recall rate of our classifier is significantly higher than a random choice (1/14~0.071). Our accuracy, defined as the ratio of the sum of TP (True Positive) and TN (True Negative) over the sum of TP, TN, FP (False Positive), and FN (False Negative), for various folds for the first experiment, is shown in Table 6.

Our classifier outperformed other directly comparable classifiers in previous works, as shown in Table 7. In particular, Fitzgerald et al. [15] obtained an average prediction accuracy of 49.1%. Moreover, Axelsson [5] was able to achieve a prediction accuracy of 34% for 28 file types. Additionally, Veenman was able to achieve a prediction accuracy of 45% for 11 file types. Xu [16] used 29 file types and obtained an average accuracy of 54.7%. For the fourteen file types in our study, the accuracy for each file type is shown in Table 7. On average, for the 14 file types which we used, the average accuracy of detection was highest for Fitzgerald at 57.51%. These researchers used the same Digital Corpora, known as govdocs1 corpus, with random sampling.

Table 6. Accuracy of 10 folds for C = 75, gamma = 0.0001.

Fold	Accuracy
1	68.57%
2	67.78%
3	67.42%
4	67.68%
5	69.14%
6	67.14%
7	67.42%
8	66.78%
9	66.64%
10	66.21%

Table 7. Average accuracies of some previous works for the 14 file types in our study %.

FileType	Hier.Class (Our Approach)	Fitz [15]	Xu [16]	Axelsson [5]
csv	91.5	99.7	39.0	33.3
doc	53.0	58.0	25.0	25.3
gif	79.0	95.8	45.0	0.7
gz	48.0	24.8	4.0	0.7
html	42.0	94.8	34.0	14.2
jpg	71.0	17.4	22.0	0.7
pdf	70.0	29.2	22.0	10.7
png	71.75	62.5	15.0	0.7
ppt	48.3	13.6	11.0	0.7
ps	66.21	98.5	57.0	2.4
swf	71.5	14.8	21.0	0.7
txt	74.5	31.8	48.0	2.5
xls	83.2	73.5	64.0	23.4
xml	79.0	90.8	77.0	25.2
Average Values	67.78	57.51	34.57	10.09

The confusion matrix for our evaluation experiments is shown in Figures 5 and 6. Each file type had 1000 fragments in the data. As we can see from the confusion matrix, the classifier performs well on files that are simple file fragments (e.g., plain text files, png files) and worse on complex files (e.g., pdf, ppt, and doc). Additionally, from Tables 3 and 4, we found that the *TPR* (True Positive Rate) rates for these file types were lower compared to other file types.

For simple files, our proposed method achieves better results for xml (Table 4) files compared to html files in terms of F1-measure values. The F1-measure values of xml and html files are 0.85 vs. 0.71, respectively. This is because xml defines data separately from their presentation, which makes xml data easier to locate and manipulate. The xml file mainly focuses on the transfer of data, while html focuses on the presentation of the data. The higher heterogeneity of contents that usually exist in html files also makes it difficult to classify html files. For other simple files, namely jpg and png, the classifier achieves average F1-measure values of 0.57 and 0.59, respectively. This result is due to the fact both jpg and png use compression methods (DCT (Discrete Cosine Transform) and LZW (Lempel-Ziv-Welch)) to minimize the size of the image.

Upon examination of the pdf files, we found that the file fragments belonging to the pdfs that were misclassified consisted of a large portion of images. Table 8 shows the percentage of complex files that contain the image in our dataset. About 90.76% of ppt files and 58.30% of pdf files contain image, while the F1-measures for ppt and pdf files are 0.14 and 0.19, respectively. For doc files, only 3.07% of files contained image and the F1-measure for the doc file is 0.51. Therefore, it is evident that images in complex files have a significant impact on the file fragment classification.

Table 8. Complex files that contain images.

Complex File Types	No. of Files	Files Containing Image	Percentage
PPT	4092	3714	90.76%
PDF	18,610	10,817	58.30%
DOC	5764	177	3.07%

Moreover, the pdf files usually store an image as a separate object that contains the raw binary data of the image. Images are not embedded inside a pdf as tif, gif, bmp, jpeg, or png formats. The image bytes are modified when the pdf is created and different pdf creation tools may store the same image in very different ways. After analyzing the metadata of pdf files in our dataset, we found that the pdfs are created by 573 different pdf creator software programs (including different versions). Moreover, the pdf creator software produce different versions of pdf files. This suggests that our hierarchy needs fine-tuning with regards to complex files, as currently all complex files are placed under the same parent node.

7. Conclusions and Future Works

File fragment classification is of paramount importance in digital forensics. In this work, we explored a classification approach called hierarchical classification for file fragment classification. Owing to its hierarchical nature, if a correct taxonomy is found we believe that this could be a general technique used for fragment classification. Here, we used support vector machines (SVMs) as our base classifiers and also proposed a hierarchical taxonomy for files, such that if a new file is introduced and this technique needs to be used then this is made possible. Furthermore, we investigated the effects of the hyperparameters *C* and *gamma* and found that the parameters best suited to our model had values of $C = 75$ and $gamma = 0.0001$. Then, we also tested our classification technique using a dataset consisting of 14 file types and with 1000 fragments, each having 512-byte fragments for each file type, via 10-fold cross-validation technique. Furthermore, we refined our first hierarchy slightly and conducted the same experiment on the same dataset. We compared the F1 measures for both our experiment runs to see if refining our hierarchy would help in the classification process.

Our classifier either outperformed or gave comparable results to all preceding similar classifiers suggested in the literature. From the confusion matrix presented in Figures 5 and 6, we discovered that our classifier worked best for simple file types and worst for complex file types. On further examination, it was found that some of the file fragments in the pdf category that were incorrectly classified consisted of a lot of pictures. We believe that the features utilized in this study are not adequate, as none of our features characterize the membership relationship between the file fragment and its parent class, especially for complex file types. Moreover, after refining the simple file branch in terms of the hierarchical complexity, we found that our classifier performed better for complex file types. This suggests that we need to further fine-tune our hierarchy branch for complex files or completely eliminate it and make our hierarchy completely file-content-based. Furthermore, it was interesting that most of the incorrectly classified file fragments belonging to the jpg file type were classified as png file fragments. This suggests that a deeper look needs to be taken towards engineering other features that can characterize the membership relationship of a file fragment with its parent file type.

The results of the hierarchical classification approach to file fragment classification problems are quite promising, and we believe that our approach has provided a different perspective (i.e., a hierarchical perspective) for looking at and solving this complex problem. Here, we can see several areas of research in the future. Firstly, we believe that a fuzzy metric should be developed and a membership function should be defined to characterize the membership of complex and simple file categories. This would eliminate the need for having a complex file type branch, and these files could be represented merely by the superposition of simple file types. Secondly, the hierarchy we proposed needs to be further tuned, such that the misclassifications can be eliminated. Additionally,

encrypted files need to be introduced into the hierarchy somehow. Research should be directed toward finding a way to include standard pattern-matching-based fragment classification and the hierarchical classification technique. Pattern matching could be included as a feature in the classification process. Thirdly, the effects of different pdf creation tools and pdf versions should be further analyzed. Additionally, the govdocs1 corpus does not contain zip files or recent Microsoft Office formats (i.e., docx, xlsx, pptx). These new file formats also need to be analyzed. Moreover, new Microsoft Office formats (docx, xlsx, pptx) use file archiving computer program, named PKZIP (written by Phil Katz), to compress content, so it will be interesting to see the comparison of both old and recent Microsoft Office formats with other compressed file formats (i.e., gz and zip). Another interesting research topic could be the analysis of the effects of the defragmentation tool. Usually, files are stored in different sectors in a disk as fragments, which are not contiguous. Defragmentation tools are used to move these file fragments to make them contiguous. Finally, the use of other classification techniques as base classifiers needs to be investigated, such that researchers can choose the classifier that best suits their needs and interests.

Author Contributions: M.B., A.M., and M.W.U.K. contributed equally to this work for data-collection, coding, and execution. S.E.B.-G. and R.R. helped data curation and writeup. M.T.H. and I.A. designed the overall experiments, wrote and reviewed the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was in part supported by the NSF grant # 1623276 and Louisiana Board of Regents Support Fund LEQSF-(2016-19)-RD-B-07.

Acknowledgments: The author would like to acknowledge the constructive suggestions of the anonymous reviewers.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Casey, E.; Rose, C.W. Chapter 2—Forensic Analysis. In *Handbook of Digital Forensics and Investigation*; Casey, E., Altheide, C., Daywalt, C., de Donno, A., Forte, D., Holley, J.O., Johnston, A., van der Knijff, R., Kokocinski, A., Luehr, P.H., et al., Eds.; Academic Press: San Diego, CA, USA, 2010; pp. 21–62. [CrossRef]
2. Roussev, V.; Garfinkel, S.L. File Fragment Classification—The Case for Specialized Approaches. In Proceedings of the 2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, Berkeley, CA, USA, 21 May 2009; pp. 3–14.
3. Roussev, V.; Quates, C. File fragment encoding classification—An empirical approach. *Digit. Investig.* **2013**, *10*, S69–S77. [CrossRef]
4. Darwin, I.F. Libmagic. 2008. Available online: ftp://ftp.astron.com/pub/file/ (accessed on 2 August 2020).
5. Axelsson, S. The Normalised Compression Distance As a File Fragment Classifier. *Digit. Investig.* **2010**, *7*, S24–S31. [CrossRef]
6. Conti, G.; Bratus, S.; Shubina, A.; Sangster, B.; Ragsdale, R.; Supan, M.; Lichtenberg, A.; Perez-Alemany, R. Automated mapping of large binary objects using primitive fragment type classification. *Digit. Investig.* **2010**, *7*, S3–S12. [CrossRef]
7. Ahmed, I.; Lhee, K. Detection of Malcodes by Packet Classification. In Proceedings of the 2008 Third International Conference on Availability, Reliability and Security(ARES), Barcelona, Spain, 4–7 March 2008; pp. 1028–1035.
8. Ahmed, I.; Lhee, K.-S.; Shin, H.; Hong, M. On Improving the Accuracy and Performance of Content-Based File Type Identification. In *Information Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 44–59.
9. Ahmed, I.; Lhee, K.-S.; Shin, H.; Hong, M. Fast File-type Identification. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 1601–1602.
10. Ahmed, I.; Lhee, K.-S.; Shin, H.-J.; Hong, M.-P. Fast Content-Based File Type Identification. In *Advances in Digital Forensics VII*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 65–75.
11. Ahmed, I.; Lhee, K.-S.; Shin, H.; Hong, M. Content-Based File-Type Identification Using Cosine Similarity and a Divide-and-Conquer Approach. *IETE Tech. Rev.* **2010**, *27*, 465–477. [CrossRef]

12. Ahmed, I.; Lhee, K.-S. Classification of packet contents for malware detection. *J. Comput. Virol.* **2011**, *7*, 279. [CrossRef]
13. Li, W.-J.; Wang, K.; Stolfo, S.J.; Herzog, B. Fileprints: Identifying file types by n-gram analysis. In Proceedings of the Sixth Annual IEEE SMC Information Assurance Workshop, West Point, NY, USA, 15–17 June 2005; pp. 64–71.
14. Calhoun, W.C.; Coles, D. Predicting the types of file fragments. *Digit. Investig.* **2008**, *5*, S14–S20. [CrossRef]
15. Fitzgerald, S.; Mathews, G.; Morris, C.; Zhulyn, O. Using NLP techniques for file fragment classification. *Digit. Investig.* **2012**, *9*, S44–S49. [CrossRef]
16. Xu, T.; Xu, M.; Ren, Y.; Xu, J.; Zhang, H.; Zheng, N. A File Fragment Classification Method Based on Grayscale Image. *J. Comput.* **2014**, *9*, 1863–1870. [CrossRef]
17. Dumais, S.; Chen, H. Hierarchical classification of Web content. In Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, 24—28 July 2000; pp. 256–263.
18. Sun, A.; Lim, E.-P. Hierarchical text classification and evaluation. In Proceedings of the 2011 IEEE International Conference on Data Mining, San Jose, CA, USA, 29 November–2 December 2001; pp. 521–528.
19. Nakano, F.K.; Pinto, W.J.; Pappa, G.L.; Cerri, R. Top-down strategies for hierarchical classification of transposable elements with neural networks. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 2539–2546.
20. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
21. Vapnik, V.N. *The Nature of Statistical Learning Theory*; Springer: Berlin, Germany, 1995; p. 188.
22. Garfinkel, S.; Farrell, P.; Roussev, V.; Dinolt, G. Bringing science to digital forensics with standardized forensic corpora. *Digit. Investig.* **2009**, *6*, S2–S11. [CrossRef]
23. Rennie, J.D.M. Derivation of the F-Measure. *Other Words*. 2004, p. 1. Available online: http://qwone.com/~{}jason/writing/fmeasure.pdf (accessed on 2 August 2020).
24. McDaniel, M.; Heydari, M.H. Content based file type detection algorithms. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Big Island, HI, USA, 6–9 January 2003.
25. Karresand, M.; Shahmehri, N. File type identification of data fragments by their binary structure. In Proceedings of the 2006 IEEE Information Assurance Workshop, West Point, NY, USA, 21–23 June 2006; pp. 140–147.
26. Shannon, C.E. A note on the concept of entropy. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]
27. Veenman, C.J. Statistical disk cluster classification for file carving. In Proceedings of the Third International Symposium on Information Assurance and Security, Manchester, UK, 29–31 August 2007; pp. 393–398.
28. Van Asch, V. *Macro- and Micro-Averaged Evaluation Measures [[BASIC DRAFT]]*; CLiPS, Univercity of Antwerp: Antwerp, Belgium, 2013.
29. Wang, F.; Quach, T.; Wheeler, J.; Aimone, J.B.; James, C.D. Sparse Coding for N-Gram Feature Extraction and Training for File Fragment Classification. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 2553–2562. [CrossRef]
30. Chen, Q.; Liao, Q.; Jiang, Z.L.; Fang, J.; Yiu, S.; Xi, G.; Li, R.; Yi, Z.; Wang, X.; Hui, L.C.K.; et al. File Fragment Classification Using Grayscale Image Conversion and Deep Learning in Digital Forensics. In Proceedings of the IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 140–147.
31. Mittal, G.; Korus, P.; Memon, N. FiFTy: Large-Scale File Fragment Type Identification Using Neural Networks. *arXiv* **2019**, arXiv:1908.06148. [CrossRef]
32. Stojanova, D.; Ceci, M.; Appice, A.; Malerba, D.; Džeroski, S. Global and Local Spatial Autocorrelation in Predictive Clustering Trees. In *Discovery Science*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 307–322.
33. Borges, H.B.; Nievola, J.C. Hierarchical classification using a Competitive Neural Network. In Proceedings of the 8th International Conference on Natural Computation, Chongqing, China, 29–31 May 2012; pp. 172–177.
34. Beebe, N.; Liu, L.; Sun, M. Data Type Classification: Hierarchical Class-to-Type Modeling. In *Advances in Digital Forensics XII*; Springer: Cham, Switzerland, 2016; Volume 484, pp. 325–343.
35. Vailaya, A.; Figueiredo, M.; Jain, A.; Zhang, H.J. Content-based hierarchical classification of vacation images. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, 7–11 June 1999; pp. 518–523.
36. Dekel, O.; Keshet, J.; Singer, Y. Large margin hierarchical classification. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; pp. 27–34.

37. Arabie, P.; De Soete, G. *Clustering and Classification*; World Scientific: Singapore, 1996.
38. Cherkassky, V.; Mulier, F. *Learning from Data: Concepts, Theory, and Methods*; John Wiley & Sons: New York, NY, USA, 1998.
39. Kuchi, A.; Hoque, M.T.; Abdelguerfi, M.; Flanagin, M.C. Machine learning applications in detecting sand boils from images. *Array* **2019**, *3*, 100012. [CrossRef]