Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Using NLP techniques for file fragment classification

Simran Fitzgerald, George Mathews, Colin Morris, Oles Zhulyn*

Department of Computer Science, University of Toronto, Canada

Keywords:
File fragment classification
File carving
Natural language processing
Bigrams
Machine learning
Support vector machine
Digital forensics

A B S T R A C T

The classification of file fragments is an important problem in digital forensics. The literature does not include comprehensive work on applying machine learning techniques to this problem. In this work, we explore the use of techniques from natural language processing to classify file fragments. We take a supervised learning approach, based on the use of support vector machines combined with the bag-of-words model, where text documents are represented as unordered bags of words. This technique has been repeatedly shown to be effective and robust in classifying text documents (e.g., in distinguishing positive movie reviews from negative ones).

In our approach, we represent file fragments as "bags of bytes" with feature vectors consisting of unigram and bigram counts, as well as other statistical measurements (including entropy and others). We made use of the publicly available Garfinkel data corpus to generate file fragments for training and testing. We ran a series of experiments, and found that this approach is effective in this domain as well.

1. Introduction

The classification of file fragments is an important problem in digital forensics, particularly for the purpose of carving fragmented files. Files that are not stored contiguously on the hard drive must be carefully reconstructed from fragments based on their content during file carving. Because the search space for fragments belonging to a particular file is so large, it is essential to have an automated method for distinguishing whether a fragment potentially belongs to a file or not. For example, a fragment from a plain-text file (e.g. `txt`) certainly does not belong to the non-metadata portion of a compressed image file (e.g. `jpg`). Garfinkel observes that although the fragmentation of files is relatively rare on today's file systems, there are cases where it commonly occurs, such as when large files are modified numerous times over a long period of time on a hard drive that is filled near capacity (Garfinkel, 2007). In this work, we explore the application of machine learning techniques from natural language processing to the problem of file fragment classification.

Classification is a standard machine learning problem. There has been work done on applying machine learning techniques to the problem of file fragment classification (Axelsson, 2010; Conti et al., 2010; Li et al., 2010; Veenman, 2007). However, the body of work in the literature is not exhaustive. Our contribution to this problem is the application of supervised machine learning techniques used in natural language processing.

In previous work, the histograms of the bytes within file fragments are used for classification (Li et al., 2005, 2010; Stolfo et al., 2005; Veenman, 2007). In natural language processing, this kind of approach is called the "bag-of-words model", where text documents are represented as unordered bags of words. A word is text separated by whitespace. The single word tokens are called unigrams, but tokens consisting of any fixed number of words can also be considered. Two consecutive word tokens are called bigrams, and bigram counts capture more information about the structure of the data being classified than do unigram counts alone. Combined with various machine

* Corresponding author.
E-mail addresses: simran.fitzgerald@utoronto.ca (S. Fitzgerald), psykeron@cs.toronto.edu (G. Mathews), colin@cs.toronto.edu (C. Morris), oles@cs.toronto.edu (O. Zhulyn).

1742-2876/$ – see front matter © 2012 O. Zhulyn, S. Fitzgerald & G. Mathews. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2012.05.008

learning techniques, this kind of approach has been repeatedly shown to be effective and robust in classifying text documents (e.g. in determining whether a piece of text has a positive or negative sentiment). In our "bag-of-bytes" approach, we treat individual bytes as tokens. We consider the unigram and bigram counts of the bytes within file fragments, along with other statistical measurements, to generate feature vector representations of the file fragments, which we then classify based on 24 different file types using a support vector machine.

Support vector machines are supervised machine learning algorithms that are very effective for classification problems (Li et al., 2010). During the training phase, a support vector machine partitions a high-dimensional space based on the points it contains that belong to known classes. File fragments can be represented in the high-dimensional space by being transformed into feature vectors. During the testing phase, file fragments of unknown types are transformed into feature vectors and are classified according to what partition they lie in in the high-dimensional space. We make use of the `libsvm` (Chang and Lin, 2011) library, which is the one of the most widely used implementations of support vector machines, to perform our experiments.

Most of the previous work on this problem exclusively uses private data sets, making it more difficult for other researchers to reproduce experimental results. We follow the example of Axelsson (2010) and derive the data set we use for training and testing from the freely available corpus of forensics research data by Garfinkel et al. (2009) (the `govdocs1` data set described in Section 4 of the cited paper). We determined the most well-represented file types in the data set and selected 24 of them based on how well-known they are. For each of the 24 file types supported by our classifier, we downloaded files uniformly at random from the `govdocs1` data set such that we would have at least 10 files made up of at least 10000 512-byte fragments. From this, we uniformly at random selected 9000 fragments for each file type to create our data set. When generating the fragments, we omitted the first and last fragments of each file, as the first fragment frequently contains header information that identifies the file type, and the last fragment might not be 512 bytes in length.

Our experiments consisted of selecting uniformly at random the same fixed number of file fragments for each of the 24 file types. We partitioned the resulting set of file fragments into a training set and a testing set in a 9-to-1 ratio, such that no file had file fragments appearing in both sets. We trained the support vector machine on the training set and tested it on the testing set. The results we got are very promising, outperforming comparable results from previous work.

The paper is organized as follows. In Section 2, we provide a brief overview of related work done in this area. In Section 3, we describe our experimental setup, which includes how we generated the data set we used in training and testing, as well as details about the features we used in our feature vectors. In Section 4, we present our results. We conclude and suggest future directions for this work in Section 5.

2. Related work

Previous work that explores the application of machine learning techniques to the problem of file fragment classification appears in the literature.

Calhoun and Coles (2008) considered only four file types (namely, `jpg`, `bmp`, `gif`, and `pdf`). For each pair of these file types, they used linear discriminant analysis (Fisher, 1936) (which is used to find linear combinations of features to characterize or separate objects between classes) in order to classify a file fragment as having one type or the other. The features they considered were various statistical measurements, including Shannon entropy (Shannon, 1948) and frequency of ASCII codes. They achieved fairly good accuracy (88.3%). However, since they classified fragments based on only four file types on a pairwise basis, it is not clear how well this technique would generalize to a real-world application, where a given file fragment is not known to belong to a file of only two possible types.

Axelsson (2010) considered 28 different file types and applied the k-nearest-neighbors classification technique with nearest compression distance as the distance metric between file fragments. The file fragment data was generated from the freely available `govdocs1` corpus by Garfinkel et al. (2009). Axelsson's experiments consisted of ten trials. In each trial, 10 files were selected at random (with the types of the files uniformly distributed) and 14 512-byte fragments were extracted at random from each of them. The fragments were then classified against a data set of approximately 3000 file fragments with known types. The average classification accuracy was around 34%, with higher accuracy being achieved for file fragments with lower entropy.

Conti et al. (2010) made use of a private data set of 14000 1024-byte binary fragments which they characterized by vectors of statistical measurements (namely, Shannon entropy, Hamming weight, Chi-square goodness-of-fit, and mean byte value). They classified each of these vectors against the remaining 13999 using k-nearest-neighbors with Euclidean distance as the distance metric. They achieved 98.55% classification accuracy for Random/Compressed/Encrypted fragments, 100% for Base64 Encoded fragments, 100% for Unencoded fragments, 96.7% for Machine Code (ELF and PE) fragments, 98.7% for Text fragments, and 82.5% for Bitmap fragments. However, the classifier did not perform as well when applied to real-world binary data, especially when it contained fragments of "a previously unstudied primitive type, even one with a closely related structure". They also classified the fragments according to types at a very coarse granularity.

Li et al. (2005) used the histogram of the byte values (unigram counts) of the prefix of a file (along with other portions of the file) in order to classify its type. They first collected a private data set of files across 8 different file types, and applied the k-means clustering algorithm to generate models for each file type (i.e. the centroids for the histograms of the files of that type). They achieved very good classification accuracy. However, this approach explicitly relies on the header information contained in each file, and hence, is not applicable to most file fragments which do not contain this information.

Veenman (2007) used the histogram of the byte values (unigram counts), the Shannon entropy, and the algorithmic or Kolmogorov complexity (Kolmogorov, 1965; Lempel and Ziv, 1976) as features for linear discriminant analysis to classify file fragments that were 4096 bytes in size. Veenman used a large private data set consisting of between 3000 and 20000 fragments per file type, for 11 file types. Veenman achieved an average classification accuracy of 45%.

Li et al. (2010) used a support vector machine with feature vectors based on the histogram of the byte values (unigram counts) to classify high entropy file fragments that were 4096 bytes in size. They used a private data set consisting of 880 `jpg` images, 880 `mp3` music files, 880 `pdf` documents, and 880 `dll` files for training and testing the support vector machine. They achieved an average classification accuracy of 81.5%. It is likely that the large file fragment size made the classification task easier. However, in the absence of file system information in a real-world situation, a large file fragment size cannot be assumed. Furthermore, this work does not take into consideration other high entropy file types that might be of interest (such as `zip` or `gz` compressed files). Some differences between the work of Li et al. and ours are as follows. The data set we used is derived from a freely available corpus, making it easier to reproduce our work, unlike Li et al. We considered a file fragment size which can be safely assumed when no file system information is available (Axelsson, 2010), unlike Li et al. Our classifier supports a much larger variety of file types, including both low and high entropy ones, unlike Li et al.

Gopal et al. (2011) considered the general problem of file classification. They defined multiple file corruption scenarios, where Type3 corresponds to file fragmentation. They then evaluated the performance of several statistical classification methods (support vector machines with n-gram feature vectors, and k-nearest-neighbors with cosine similarity as the distance metric) and several commercial off-the-shelf solutions (including Libmagic, TrID, Outside-In, and DROID) in classifying files under the various corruption scenarios. They made use of the freely available RealisticDC corpus by Garfinkel et al. (2009) in their experiments. They found that file fragment classification is a more difficult problem than file classification. Since their focus is not file fragment classification, Gopal et al. do not address several issues that are important in this problem. They mentioned that the data set they derived from the RealisticDC corpus consisted of 31644 files with 316 unique file-type extensions, where 213 of these were binary file types and 103 were ASCII text file types. However, they did not report which of these file types were considered during the experiments, nor what the proportion was of file fragments with binary file types to file fragments with ASCII-text file types in the data set used in the experiments. They also did not report how the classifiers performed on each file type. Although they made sure that the training and testing data sets did not overlap, it is not clear whether they ensured that file fragments from the same file did not appear in both sets (their results may be biased if they did not). It is also not clear whether they omitted the first file fragment of each file, which typically contains file-type revealing header information. All of these issues are addressed in our paper. Although Gopal et al. looked at file classification using k-nearest-neighbors and the various commercial off-the-shelf solutions, they did not report the performance of any of these on file fragment classification (nor is it clear that these experiments were even performed). The only results reported by Gopal et al. with respect to file fragment classification are those using the support vector machine with bigram feature vectors. They did not consider unigrams and bigrams in combination, as we do. The classification accuracy they achieved on 512-byte file fragments is about 33% using the macro-averaged F_1 measure (Özgür et al., 2005). The performance we achieved using this measure is greater.

3. Experimental setup

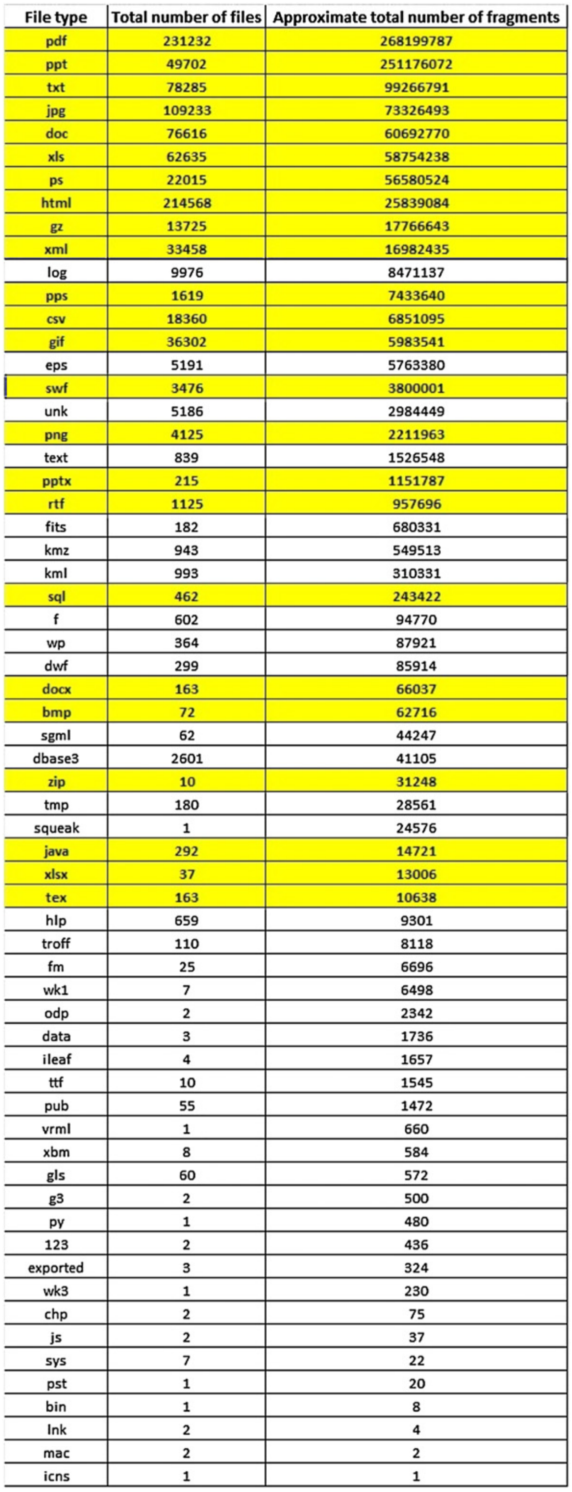
3.1. Data set

The data set we used for training and testing is derived from the freely available corpus of forensics research data by Garfinkel et al. (2009) (the `govdocs1` data set described in Section 4 of the cited paper). We determined the most well-represented file types in the data set and selected 24 of them based on how well-known they are to us (see Fig. 1). The first of these criteria ensured that we acquired a good variety of file fragment data for each file type. The second of these criteria is not a rigorous one. Although we aimed to get a good representation of file types that are likely to be of forensic interest, a rigorous methodology for selecting the most appropriate file types is outside the scope of this paper. Nevertheless, most of the file types we selected overlap with the ones selected by Axelsson (2010) who made use of the same Garfinkel corpus to derive his data set.

After selecting the file types, we proceeded to download files uniformly at random from the `govdocs1` corpus such that we would have at least 10 files made up of at least 10000 512-byte fragments for each of the 24 file types. Because the files in the `govdocs1` corpus are of variable length, and files from different file types are not equally represented, it was necessary to download more than 10 files or files that altogether constituted more than 10000 fragments, for each of the file types, in order to meet both criteria. From this, we uniformly at random selected 9000 fragments for each file type to create our data set. When generating the fragments, we omitted the first and last fragments of each file, as the first fragment frequently contains header information that identifies the file type, and the last fragment might not be 512 bytes in length. Calhoun and Coles (2008), Conti et al. (2010), and Li et al. (2010) omit these fragments as well. This approach enabled us to generate a large data set of file fragments with an equal number of file fragments for each file type, and with each fragment being derived from a variety of files with the same type.

3.2. Feature vectors

During the training phase, a support vector machine partitions a high-dimensional space based on data points



File type	Total number of files	Approximate total number of fragments
pdf	231232	268199787
ppt	49702	251176072
txt	78285	99266791
jpg	109233	73326493
doc	76616	60692770
xls	62635	58754238
ps	22015	56580524
html	214568	25839084
gz	13725	17766643
xml	33458	16982435
log	9976	8471137
pps	1619	7433640
csv	18360	6851095
gif	36302	5983541
eps	5191	5763380
swf	3476	3800001
unk	5186	2984449
png	4125	2211963
text	839	1526548
pptx	215	1151787
rtf	1125	957696
fits	182	680331
kmz	943	549513
kml	993	310331
sql	462	243422
f	602	94770
wp	364	87921
dwf	299	85914
docx	163	66037
bmp	72	62716
sgml	62	44247
dbase3	2601	41105
zip	10	31248
tmp	180	28561
squeak	1	24576
java	292	14721
xlsx	37	13006
tex	163	10638
hlp	659	9301
troff	110	8118
fm	25	6696
wk1	7	6498
odp	2	2342
data	3	1736
ileaf	4	1657
ttf	10	1545
pub	55	1472
vrml	1	660
xbm	8	584
gls	60	572
g3	2	500
py	1	480
123	2	436
exported	3	324
wk3	1	230
chp	2	75
js	2	37
sys	7	22
pst	1	20
bin	1	8
lnk	2	4
mac	2	2
icns	1	1

Fig. 1. Statistics on the `govdocs1` data set. The highlighted file types are the ones we used for our classifier.

with known classes, with each partition corresponding to a class. In order to train a support vector machine on the file fragment data, it is necessary to represent each file fragment as a point in a high-dimensional space. We do this by transforming each file fragment into a vector of features. The features we used are described here.

There are 256 features which are the histogram of the byte values (i.e. the unigram counts) for the file fragment. The feature vectors in the work by Li et al. (2010) consist only of the unigram counts. Another 256^2 features in our work are the histogram of the pairs of consecutive byte values (i.e. the bigram counts) for the file fragment.

We also have features for various statistical measurements. We have the Shannon entropy (Shannon, 1948) of the bigram counts. Conti et al. (2010) considered the Shannon entropy of the unigram, bigram, and trigram counts, and found that the entropy of the bigram counts was effective for classifying file fragments. We made use of the Hamming weight (the total number of ones divided by the total number of bits in the file fragment) and the mean byte value features that appear in the paper by Conti et al. We also used the compressed length of the file fragment as a feature. We did this to approximate the algorithmic or Kolmogorov complexity of the file fragment, following Veenman (2007). We compressed each file fragment with the `bzip2` algorithm (Seward, 2001). We also used two features from natural language processing. We computed the average contiguity between bytes (i.e. the average distance between consecutive bytes) for each file fragment, which is defined as follows:

$$\sum_{i=0}^{n=510} \frac{|fragment[i] - fragment[i+1]|}{511}$$

where *fragment* $[i]$ is the ith byte of the file fragment. Last of all, we calculated the longest contiguous streak of repeating bytes for each file fragment.

3.3. *Experiments*

We ran three experiments. For each experiment, we selected uniformly at random file fragments for each of the 24 file types. The only difference between the three experiments was the number of file fragments for each file type that was selected (i.e. 1000, 2000, and 4000). During an experiment, the set of file fragments was partitioned into a training set and a testing set in a, roughly, 9-to-1 ratio. We ensured that no file had file fragments occurring in both the training set and the testing set, so that our results would not be biased. This is the reason that the ratio was “roughly” 9-to-1. Since the set of file fragments was selected at random, we could not always achieve an exactly 9-to-1 ratio.

We then proceeded to train the support vector machine on the training set with the linear kernel. Li et al. (2010) found that the linear kernel was the most effective when classifying file fragments represented by feature vectors consisting of the histogram of the byte values (unigram counts). During some preliminary experiments, we also trained the support vector machine using the radial basis function as the kernel, but we found that it was not as effective as the linear kernel, which supports the findings by Li et al. We used the default value of 1 for the *C* parameter. Although the optimal value would improve the prediction accuracy of our classifier, we achieved good results with the default value.

Although the libsvm documentation (Chang and Lin, 2011) recommends that scaling be performed on the feature vectors to decrease training time and improve classification accuracy, we had some technical issues with the scaling program provided with libsvm which we could not resolve. As such, we did not perform scaling on our feature vectors. That being said, we were still able to achieve good results.

Once a model was generated from the training data, we had the support vector machine attempt to classify the file fragments in the testing set. We validated our results by repeating each of the three experiments ten times, each time selecting file fragments uniformly at random from our data set.

4. Results

After running the experiments, we found that our approach produced very good results. For the experiments with 1000 file fragments per file type, we achieved an average prediction accuracy of 47.5% (as reported by libsvm) with a standard deviation of 1.56%, over the ten runs. Using the macro-averaged F_1 metric, we averaged 46.3% with a standard deviation of 1.65%, over the ten runs. For the experiments with 2000 file fragments per file type, we achieved an average prediction accuracy of 48.3% (as reported by libsvm) with a standard deviation of 3.11%, over the ten runs. Using the macro-averaged F_1 metric, we averaged 47.6% with a standard deviation of 2.00%, over the ten runs. For the experiments with 4000 file fragments per file type, we achieved an average prediction accuracy of 49.1% (as reported by libsvm) with a standard deviation of 3.15%, over the ten runs. Using the macro-averaged F_1 metric, we averaged 48.0% with a standard deviation of 2.02%, over the ten runs. The prediction accuracy increased as the number of file fragments per file type increased, albeit not proportionally.

The prediction accuracy of our classifier is significantly better than random chance ($1/24 \approx 4.17\%$). Our classifier has also outperformed the most directly comparable classifiers in previous work. In particular, recall that Axelsson (2010) achieved an average prediction accuracy of 34% on 28 file types, and Veenman (2007) achieved an average prediction accuracy of 45%, but only on 11 file types. Our classifier achieved an average prediction accuracy of 49.1% on 24 file types. Gopal et al. (2011) achieved a prediction

File type	Prediction accuracy
csv	99.7%
ps	98.5%
gif	95.8%
sql	94.9%
html	94.8%
java	94.7%
xml	90.8%
bmp	90.6%
tex	87.8%
xls	73.5%
png	62.5%
doc	58.0%
rtf	38.1%
txt	31.8%
pdf	29.2%
gz	24.8%
zip	18.8%
jpg	17.4%
swf	14.8%
ppt	13.6%
xlsx	6.8%
docx	5.5%
pps	4.7%
pptx	2.3%

Fig. 2. File types ranked according to prediction accuracy for one run of the experiment with 4000 file fragments per file type.

	zip	jpg	gz	png	swf	pptx	docx	xlsx	pps	ppt	doc	pdf	txt	rtf	html	xml	xls	gif	ps	csv	sql	java	tex	bmp
zip	18.8%	3.1%	20.5%	16.7%	6.1%	0.3%	15.7%	3.4%	0.3%	1.7%	2.4%	8.9%	0.0%	0.0%	0.0%	0.0%	0.0%	2.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
jpg	9.5%	17.4%	1.7%	8.3%	29.9%	0.4%	16.6%	0.4%	6.2%	2.9%	1.2%	5.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
gz	15.2%	2.7%	24.8%	22.7%	14.8%	1.0%	3.3%	2.9%	0.8%	0.0%	1.0%	8.6%	0.0%	0.0%	0.0%	0.0%	0.0%	2.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
png	9.7%	0.6%	8.0%	62.5%	11.5%	1.4%	0.3%	1.4%	2.0%	0.0%	0.3%	2.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
swf	9.8%	3.9%	6.0%	14.9%	14.8%	1.4%	5.9%	5.5%	5.7%	2.7%	0.5%	3.2%	0.0%	0.0%	0.0%	0.0%	1.1%	23.1%	1.4%	0.0%	0.0%	0.0%	0.0%	0.0%
pptx	29.3%	0.4%	4.2%	8.5%	15.8%	2.3%	5.8%	7.3%	3.9%	0.4%	0.0%	20.5%	0.0%	0.0%	0.0%	0.0%	0.0%	1.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
docx	5.9%	4.3%	5.1%	7.8%	12.9%	50.4%	5.5%	1.6%	0.6%	0.4%	2.7%	1.2%	0.0%	0.0%	0.0%	0.2%	0.0%	0.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.6%
xlsx	19.2%	2.5%	13.9%	14.2%	14.9%	1.8%	5.7%	6.8%	2.8%	0.4%	1.4%	8.5%	0.0%	0.0%	5.7%	0.0%	0.0%	2.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
pps	9.5%	10.8%	0.9%	0.4%	48.4%	8.5%	0.3%	0.1%	4.7%	2.3%	13.1%	0.1%	0.0%	0.0%	0.0%	0.0%	0.3%	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.6%
ppt	12.6%	1.6%	15.5%	16.8%	9.4%	0.3%	3.5%	1.6%	9.9%	13.6%	3.7%	6.1%	0.0%	0.0%	0.0%	0.0%	2.4%	0.0%	0.5%	0.0%	0.3%	0.0%	0.0%	2.1%
doc	1.5%	0.7%	1.9%	1.1%	2.6%	0.0%	1.1%	1.5%	4.5%	3.3%	58.0%	1.5%	2.2%	0.0%	6.3%	1.5%	10.8%	0.0%	0.0%	0.0%	0.0%	0.4%	0.0%	1.1%
pdf	5.4%	0.0%	22.9%	15.8%	8.3%	0.0%	1.3%	0.4%	0.0%	0.0%	0.4%	29.2%	0.0%	0.0%	0.4%	0.0%	0.0%	0.8%	15.0%	0.0%	0.0%	0.0%	0.0%	0.0%
txt	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	18.2%	0.0%	31.8%	0.0%	36.1%	0.7%	6.2%	0.0%	3.6%	0.0%	0.0%	0.0%	3.3%	0.0%
rtf	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.0%	38.1%	59.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.5%	0.0%
html	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.5%	0.0%	0.3%	0.0%	94.8%	1.0%	0.0%	0.0%	0.0%	0.0%	1.0%	2.3%	0.0%	0.0%
xml	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.8%	0.0%	7.8%	90.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.3%	0.0%
xls	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4.9%	0.4%	1.1%	18.3%	0.0%	0.0%	0.0%	0.0%	0.0%	73.5%	0.0%	0.0%	0.0%	1.5%	0.0%	0.0%	0.4%
gif	0.0%	0.3%	0.3%	2.0%	0.5%	0.0%	0.2%	0.2%	0.0%	0.0%	0.0%	0.7%	0.0%	0.0%	0.0%	0.0%	0.0%	95.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
ps	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.0%	0.0%	0.0%	0.3%	0.0%	0.0%	0.0%	98.5%	0.0%	0.0%	0.0%	0.3%	0.0%
csv	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	99.7%	0.0%	0.0%	0.0%	0.0%
sql	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.4%	0.0%	1.6%	0.8%	0.0%	0.0%	1.6%	0.0%	94.9%	0.4%	0.3%	0.0%
java	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.0%	4.1%	0.0%	0.0%	0.0%	0.0%	0.6%	0.0%	94.7%	0.3%	0.0%
tex	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%	0.0%	0.0%	0.0%	11.0%	0.0%	0.0%	0.3%	87.8%	0.0%
bmp	0.5%	0.2%	0.0%	0.0%	4.8%	0.6%	0.3%	0.0%	0.6%	0.0%	0.3%	0.0%	0.0%	0.0%	0.2%	0.0%	1.0%	0.0%	1.0%	0.0%	0.0%	0.0%	0.0%	90.6%

Fig. 3. Confusion matrix for one run of the experiment with 4000 file fragments per file type. The rows correspond to the actual file types and the columns correspond to what file types the file fragments were classified as. The file types in the top left correspond to high entropy files.

accuracy of about 33% using the macro-averaged F_1 metric. Using the same metric, our average prediction accuracy was 48.0%.

Our classifier performed best on low entropy file fragments (e.g. plain-text files, uncompressed images, etc.), and worst on high entropy file fragments (e.g. compressed files, binary executables, etc.) (see Figs. 2 and 3). This reflects the findings in previous work (i.e. high entropy file fragments are harder to classify). In future work, the improvement of classification performance on high entropy file fragments should be investigated, such as through the use of the various machine learning boosting techniques.

The goal of this project was to investigate whether techniques from natural language processing could be applied successfully to file fragment classification. We found that this is indeed the case. However, it is not clear how much each of the features we considered contributed to the results we achieved, and this should be investigated in future work. In preliminary work to this end, we found that the bigram counts provided the most significant contribution. The unigram counts, contiguity, and Hamming weight were also important, albeit not to the extent of the bigram counts. Due to the early nature of this work, we do not present the results here.

5. Conclusion and future work

File fragment classification is an important problem in digital forensics. For this paper, we explored the application of supervised machine learning techniques from natural language processing to this problem. We generated a large data set of file fragments for 24 different file types, which we derived from the freely available `govdocs1` corpus by Garfinkel et al. (2009). We represented each file fragment with a feature vector consisting of the unigram and bigram counts of bytes in the fragment, along with several other statistical measurements (such as entropy). Our file fragment classification approach consisted of using a support vector machine along with these feature vectors. We ran several experiments, and found this to be an effective approach, which outperformed comparable ones in the literature.

The results we got are promising, and several directions for future work on this project are apparent. For one, by scaling the feature vectors and finding optimal parameter values for the support vector machine, both the classification accuracy and the training efficiency can be increased. Furthermore, it would be interesting to evaluate the effects the various features in our feature vectors have on the results. In particular, it is not clear whether all of the features we used were necessary to achieve the results we did. This can be done by running experiments where the feature vectors consist of only subsets of all the features we considered, and comparing the results.

It has been shown in previous work that classifying high entropy file fragments is challenging (Conti et al., 2010; Li et al., 2010), which is affirmed by our work. One possible future direction in this regard is to consider the various machine learning boosting techniques, such as AdaBoost (Freund and Schapire, 1997), which are useful for this kind of classification problem.

Finally, with the goal of an effective, general file fragment classifier in mind, it seems that a better approach might be to first use a coarsely grained but accurate classifier like the one by Conti et al. (2010) to classify a file fragment according to broad categories (e.g., Random/Compressed/Encrypted, Machine Code, Text, and Bitmap). After placing the file fragment within a category, a specialized classifier would then further classify it according to the file types within that category.

Acknowledgments

We are grateful to G. Scott Graham for suggesting the initial direction for this project and his support, to Greg Conti for his help, to the anonymous reviewers and Nicole Beebe for their comments, and to Alex Cheng for his advice.

References

Axelsson S. The normalized compression distance as a file fragment classifier. In: Proceedings of the 2010 Digital Forensics Research Conference (DFRWS); 2010.

Calhoun W, Coles D. Predicting the types of file fragments. In: Proceedings of the 2008 Digital Forensics Research Conference (DFRWS); 2008.

Chang C-C, Lin C-J. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2011;2. 27:1–27:27, software available at: http://www.csie.ntu.edu.tw/cjlin/libsvm.

Conti G, Bratus S, Sangster B, Ragsdale R, Supan M, Lichtenberg A, et al. Automated mapping of large binary objects using primitive fragment type classification. In: Proceedings of the 2010 Digital Forensics Research Conference (DFRWS); 2010.

Fisher RA. The use of multiple measurements in taxonomic problems. Annals of Eugenics 1936;7:179–88.

Freund Y, Schapire R. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 1997;55(1):119–39.

Garfinkel S. Carving contiguous and fragmented files with object validation. In: Proceedings of the 2007 Digital Forensics Research Conference (DFRWS); 2007.

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the 2009 Digital Forensics Research Conference (DFRWS); 2009.

Gopal S, Yang Y, Salomatin K, Carbonell J. Statistical learning for file-type identification. In: 2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA), Vol. 1; 2011. p. 68–73.

Kolmogorov AN. Three approaches to the quantitative definition of information. Problems of Information Transmission 1965;1:1–11.

Lempel A, Ziv J. On the complexity of finite sequences. IEEE Transactions on Information Theory 1976;22:75–81.

Li W-J, Wang K, Stolfo S, Herzog B. Fileprints: identifying file types by n-gram analysis. In: Proceedings of the 2005 IEEE workshop on information assurance and security; 2005. p. 64–71.

Li Q, Ong A, Suganthan P, Thing V. A novel support vector machine approach to high entropy data fragment classification. In: Proceedings of the South African Information Security Multi-Conference (SAISMC 2010); 2010.

Özgür A, Özgür L, Güngör T. Text categorization with class-based and corpus-based keyword selection. In: Proceedings of the 20th International Conference on Computer and Information Sciences (ISCIS'05); 2005. p. 606–15.

Seward J. Space-time tradeoffs in the inverse b-w transform. In: Proceedings of the 2001 data compression conference. IEEE Computer Society; 2001. p. 439–48.

Shannon C. A mathematical theory of communication. Bell System Technical Journal 1948;27(3,4). 379-423, 623-656.

Stolfo S, Wang K, Li W-J. Fileprint analysis for malware detection. Tech. rep.. Columbia Univesrity; June 2005

Veenman CJ. Statistical disk cluster classification for file carving. In: Proceedings of the IEEE 3rd international symposium on information assurance and security. IEEE Computer Society; 2007. p. 393–8.