Using NLP Techniques for File Fragment Classification

Abstract

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 took 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.

Keywords: file fragment classification, machine learning, support vector machine, digital forensics

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 a compressed image file (e.g. `jpg`). Garfinkel [1] observes that although the fragmentation of files is relatively rare on today's file systems, the files of interest in forensic investigations are more likely to be fragmented than other files. For example, large files that have been modified numerous times over a long period of time on a hard drive that is filled near capacity, will likely exhibit fragmentation. 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 [2, 3, 4, 5]. 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 [6, 4, 7, 5]. 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. The single word tokens are called unigrams, but tokens consisting of any fixed number of words can also be considered. Two 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 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 approach, 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 [4]. 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` [8] 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 [2] and derive the data set we use for training and testing from the freely available corpus of forensics research data by Garfinkel et al. [9] (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 [10] considered only four file types (namely, `jpg`, `bmp`, `gif`, and `pdf`). For each pair of these file types, they used linear discriminant analysis [11] (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 [12] 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 [2] 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. [9]. 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. [3] 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 of a very coarse granularity.

Li et al. [6] 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 [5] used the histogram of the byte values (unigram counts), the Shannon entropy, and the algorithmic or Kolmogorov complexity [13, 14] 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. [4] 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 [2], 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.

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

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

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

Garfinkel et al. [9] (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 Figure 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 [2] 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 [10], Conti et al. [3], and Li et al. [4] 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 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. [4] 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 [12] of the bigram counts. Conti et al. [3] 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 [5]. We compressed each file fragment with the `bzip2` algorithm [15]. We also used two features from natural language

	1000	2000	4000
Run 1	47.6311	46.5883	44.4398
Run 2	52.8807	55.9301	50.4983
Run 3	46.1976	46.9444	53.3733
Run 4	48.9196	49.0208	55.6484
Run 5	46.1818	52.0459	49.0884
Run 6	52.1587	50.9063	45.4222
Run 7	50.3359	49.9680	46.9092
Run 8	48.0272	50.0839	50.2260
Run 9	41.4222	51.5344	49.1760
Run 10	49.6409	45.7562	49.2458
Minimum	41.4222	45.7562	44.4398
Average	**48.3396**	**49.8778**	**49.4027**
Maximum	52.8807	55.9301	55.6484

Figure 2: *Classification accuracy (in percent)*

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 i^{th} 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 upper limit on 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. We then proceeded to train the support vector machine on the training set with default parameters and the linear kernel. Li et al. [4] 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). 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 repeated 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 that limited the number of file fragments per file type to 1000, we achieved an average prediction accuracy of 48.3396% (with a minimum of 41.4222% and a maximum of 52.8807% for the ten runs). For the experiments that limited the number of file fragments per file type to 2000, we achieved an average prediction accuracy of 49.8778% (with a minimum of 45.7562% and a maximum of 55.9301% for the ten runs). For the experiments that limited the number of file fragments per file type to 4000, we achieved an average prediction accuracy of 49.4027% (with a minimum of 44.4398% and a maximum of 55.6484% for the ten runs). There was no significant improvement in the average prediction accuracy for the experiments that limited the number of file fragments per file type to 4000 over the ones that limited the number to 2000.

In terms of average prediction accuracy, our classifier has outperformed the most directly comparable classifiers in previous work. In particular, recall that Axelsson [2] achieved an average prediction accuracy of 34% on 28 file types, and Veenman [5] achieved an average prediction accuracy of 45%, but only on 11 file types. Our classifier achieved an average prediction accuracy of 49.8778% on 24 file types.

Our classifier did 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 Figure 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.

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. [9]. 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 parameters 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 [3, 4], 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 [16], which are useful for this kind of classification problem.

	zip	jpg	gz	png	swf	docx	xlsx	pptx	pps	ppt	doc	pdf	txt	rtf	html	xml	xls	gif	ps	csv	sql	java	tex	bmp
zip	0.15	0.01	0.22	0.14	0.02	0.27	0.02	0.01	0	0	0.02	0.11	0	0	0	0	0	0.02	0	0	0	0	0	0
jpg	0.16	0.18	0.03	0.05	0.14	0.24	0	0.08	0.04	0.01	0.02	0.04	0	0	0	0	0	0.01	0	0	0	0	0	0
gz	0.09	0.01	0.36	0.16	0.04	0.1	0.02	0.06	0.01	0	0.03	0.1	0	0	0	0	0	0.03	0	0	0	0	0	0
png	0.06	0	0.1	0.59	0.09	0.01	0.01	0.1	0	0	0	0.02	0	0	0	0	0	0.01	0	0	0	0	0	0
swf	0.07	0.01	0.09	0.12	0.1	0.09	0.02	0.05	0.04	0.05	0.01	0.04	0	0	0	0	0.01	0.3	0	0	0	0	0	0
docx	0.07	0.02	0.09	0.08	0.01	0.12	0	0.5	0.01	0.01	0.01	0.04	0	0	0	0	0.01	0.03	0	0	0	0	0	0
xlsx	0.08	0.02	0.12	0.1	0.03	0.19	0.06	0.19	0.01	0	0.01	0.11	0	0	0.05	0	0	0.05	0	0	0	0	0	0
pptx	0.14	0.01	0.09	0.08	0.02	0.13	0.06	0.13	0.01	0	0.01	0.26	0	0	0	0	0	0.04	0	0	0	0	0	0
pps	0.11	0.01	0.19	0.19	0.02	0.08	0.01	0.09	0.02	0.1	0.06	0.07	0	0	0	0.01	0.01	0.01	0	0	0	0	0	0
ppt	0.1	0.02	0.16	0.12	0.03	0.05	0.01	0.06	0.04	0.21	0.06	0.1	0	0	0	0	0.01	0.02	0	0	0	0	0	0.02
doc	0.08	0.15	0.05	0.06	0.09	0.2	0.01	0.06	0.02	0.02	0.2	0.04	0	0	0.01	0	0.01	0.02	0	0	0	0	0	0
pdf	0.05	0	0.24	0.09	0.02	0.02	0	0.01	0	0	0	0.42	0	0	0	0	0	0	0.13	0	0	0	0	0
txt	0	0	0	0	0	0.01	0	0	0	0	0.19	0.03	0.17	0	0.32	0	0.05	0	0.05	0	0	0	0.17	0
rtf	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0.01	0	0	0	0	0	0	0	0	0
html	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0.79	0.13	0	0	0	0	0.03	0.04	0.01	0
xml	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0.26	0.73	0	0	0	0	0	0	0	0
xls	0	0	0	0	0	0	0.15	0	0	0.02	0.29	0	0	0	0	0	0.52	0	0	0	0	0	0	0.02
gif	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	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	0.99	0	0	0	0	0
csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0.15	0	0	0
sql	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0	0	0	0	0	0.97	0	0	0
java	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0	0	0	0	0	0.01	0.95	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	0.99	0
bmp	0.01	0	0	0.01	0	0	0	0.02	0	0	0	0	0	0	0	0	0.01	0	0.02	0	0	0	0	0.93

Figure 3: *Confusion matrix from one of the runs of the experiment that limited the number of file fragments per file type to 4000. 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.*

References

[1] S. Garfinkel, Carving contiguous and fragmented files with object validation, in: Proceedings of the Digital Forensics Research Conference (DFRWS) 2007, 2007.

[2] S. Axelsson, The normalized compression distance as a file fragment classifier, in: Proceedings of the Digital Forensics Research Conference (DFRWS) 2010, 2010.

[3] G. Conti, S. Bratus, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, R. Perez-Alemany, A. Shubina, Automated mapping of large binary objects using primitive fragment type classification, in: Proceedings of the Digital Forensics Research Conference (DFRWS) 2010, 2010.

[4] Q. Li, A. Ong, P. Suganthan, V. Thing, 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.

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

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

[7] S. Stolfo, K. Wang, W.-J. Li, Fileprint analysis for malware detection, Tech. rep., Columbia Univesrity (June 2005).

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

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

[10] W. Calhoun, D. Coles, Predicting the types of file fragments, in: Proceedings of the Digital Forensics Research Conference (DFRWS) 2008, 2008.

[11] R. A. Fisher, The use of multiple measurements in taxonomic problems, Annals of Eugenics 7 (1936) 179–188.

[12] C. Shannon, A mathematical theory of communication, Bell System Technical Journal 27 (3, 4) (1948) 379–423, 623–656.

[13] A. N. Kolmogorov, Three approaches to the quantitative definition of information, Problems of Information Transmission 1 (1965) 1–11.

[14] A. Lempel, J. Ziv, On the complexity of finite sequences, IEEE Transactions on Information Theory 22 (1976) 75–81.

[15] J. Seward, Space-time tradeoffs in the inverse b-w transform, in: Proceedings of the Data Compression Conference 2001, IEEE Computer Society, 2001, pp. 439–448.

[16] Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.