CarveML: application of machine learning to file fragment classification

Andrew Duffy

Stanford University
agd@cs.stanford.edu

Abstract

We present a learning algorithmic approach to the problem of recognzing the file types of file fragments, with the purpose of applying this to "file carving", the reconstruction of partially erased files on disk into whole files. We do so through the use of 257 calculated features of an input fragment, applying the Support Vector Machine, Multinomial Naive Bayes, and Linear Discriminant Analysis models to our problem to see which produces the most accurate method of classification.

I. Introduction

In the forensics community, a common scenario faced by digital forensic investigators is piecing back together useful information from a disk image. Digital forensic investigators examine these disk images in an attempt to uncover any suspicious activity, including any traces of digital materials that can be used as evidence. However, this is often challenging, as perpetrators may "delete" information off of their hard disk. These sorts of deletions, though, are not permanent: they simply mark each block of the file as unallocated and available for use in other files [1]. These unallocated file blocks can be recovered using a tool called a "file carver", an application which can take these fragments and write them back out to disk.

Most file carvers depend on header and footer information specific to certain filetypes [1]. For example, when we look at JPEG images, we know that all JPEGs begin with the byte sequence `0xFF 0xD8`, so a header-footer based carver will look for fragments that contain this pair of bytes near their start to determine that a fragment is a JPEG. While this might be a good method for classifying fragments towards the start or end of a file, it will fail for classifying the internal fragments. Because of this, it often becomes necessary to have carving methods which examine the fragment contents. Keeping in mind that different file types have different formats, and there is not much consistency between formats, we must derive some statistical similarity measures to tell how similar or different two fragments are from one another. Having derived these features, we would then apply any of our array of classification algorithms to try and train a model on our fragment data, and see how well we perform at prediction new fragments' types.

We begin our dissection of our research with a description of the dataset we used, a walkthrough of the set of derived features that we calculated per fragment, the statistical models we used to represent the data, and end with a discussion of our results and possible future work.

II. Data Set

We draw our data from the `govdocs1` corpora, a 1 million file dataset developed by Garfinkel et al. [4]. The corpus was developed by crawling through various *.gov* sites, which means the dataset is in the public domain and freely distributable. This is uncommon in the forensics field, as many researchers will use either difficult to reproduce datasets or completely private datasets [3]. Thus, performing

our research against a widely available freely distributable corpus allows anyone to compare our results to others using the same set of files.

While we are forunate to have such a large corpus, it also makes training and testing on the entire corpus unfeasible, leading to questions of data sampling. Luckily, the maintainers of the dataset have gone and created 1000 pre-packaged "threads", 1000-file randomly generated subsets of the main corpora specifically meant for further use in research. Our training thread was composed of 983 files with 28 different labels, and we tested on a set of 973 files, with one randomly chosen fragment of random size from each file, to echo the environment we might see in a real disk dump.

While we performed our testing on file fragments, for parameter estimation we trained each model on entire files. This was done with the goal in mind of creating models that could recognize whole files, and then assume that fragments we give it will have the same distributions (in general) as whole-files with the corresponding type. This method performed fairly well at allowing us to classify the random testing fragments, as we show below in the Results section.

III. Features

We did not use any direct features of the files or fragments. Rather, we calculated a 257-dimensional feature vector for each training and testing input, and used those to represent each file to our learning models.

These 257 features came from two sources. The first 256 features consist of the byte histogram, where for each of the 256 possible byte values (0-255) we had one real-valued feature that represented the proportion of fragment bytes that took on that specific value. This has been used by other researchers with strong results [3]. The other measure that we chose to extract from each input source is the Shannon entropy. The formula for calculating the Shannon entropy, H, of a file or fragment is:

$$H = -\sum_{i=0}^{255} p(x=i)\log_2 p(x=i)$$

Where $p(x=i)$ is the probability of any single byte in the fragment taking on value i. It becomes immediately clear that these probabilities are simply the byte proportions we calculate for our first 256 features, so our code for calculating Shannon entropy becomes something like:

```
def shannon_entropy(bytes):
  hist = byte_histogram(bytes) #
      Returns a dictionary of
      byte_value => probability
  sum = 0
  for i in range(256):
      sum += hist[i] * math.log(
          hist[i], 2)
  return -1*sum
```

One can think of Shannon entropy as a measure of the amount of information packed into some size of bytes. It does not correspond directly to "information per bit", but its values do correlate strongly with file type. We found in our research that for text files such as `.html`, `.txt` and `.tmp`, the calculated entropy was somewhere between 1.0 - 1.8, while for more compressed formats, including `.jpeg`, `.pdf`, and `.gz` formats, this measure was closer to 8.0. Other research used a combination of these features and others, including byte n-grams which take into account byte ordering, whereas we instead treat each input as a "bag of bytes" similar to the "bag of words" model we normally associate with text classification [3]. While this is making a very strong assumption about our dataset, it in no way hampered our ability to realize fairly strong results using just this minimum set of features.

IV. Models

For this application, because the goal was classification, we had the universe of classification methods at our disposal. As the work

in this area has been somewhat limited and no specialized algorithms were found for this topic, in this paper we only consider the use of Support Vector Machine, Mulitnomial Naive Bayes and Linear Discriminant Analysis models.

Because the research in this field is limited, and no specialized algorithms exist to solve the fragment classification problem, we chose SVMs because they are generally a good candidate for any classification problem. Naive Bayes was used to see how well we could model this problem like a text classification problem, treating each of the bytes as if they were randomly occuring words in some text stream. LDA was used just as a method of comparison to the other two, as it is also a generally strong classification algorithm. It assumes that the conditional distribution of features are distributed Gaussian, but we see in the results section that this does not harm our classification accuracy.

For the Support Vector model, we use the linear kernel, and apply L1 regularization with a parameter $C = 100$ found empirically through multiple test runs. Our classification accuracy appeared to drop for values less than 100, and for values much higher than 100 we also found decreased accuracy, so 100 was found to be optimal within a few fractions of a percent.

Also note that SVMs are not invariant to scaling [5][3], so we improve our results by clamping our features in the range of $[0, 1)$. We do this for our 256-dimensional byte histogram, possibly contributing to our strong results.

V. Results

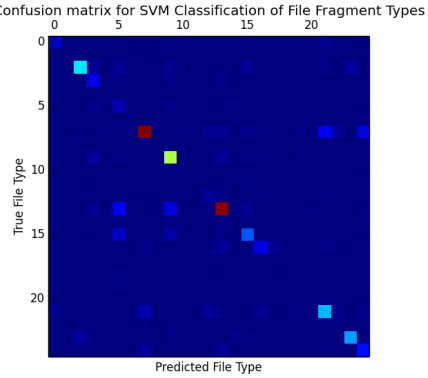
After training our models and testing against 973 distinct fragments, we found the following results, which we show in tabular form:

	Accuracy
SVM	75.03%
Multinomial Naive Bayes	47.90%
LDA	69.01%

We can see that our Support Vector Machine model had the strongest performance in our tests, exceeding maximums set by other scientific work with the same corpus [3]. It was initially surprising that the Naive Bayes model performed so much more poorly than our Support Vector Machine. However, it is likely the case that the Naive Bayes assumptions may have been too strong to correctly model our dataset, at least with our given set of features.

Meanwhile, Support Vector Machines, often the favorite for a variety of classification problems, were unsurprisingly successful in this scenario.

Here you can see the confusion matrix for the SVM. Note how the diagonal entries are the brightest, and the bright off-diagonals indicate file types that were often mis-classified as another type by the SVM (ex. HTML files were often misclassified as Python files, etc.)



While we initially only used LDA as a comparison, we can see that it performed even better than Naive Bayes, our secondary model. Considering that our features are all real-valued and can be considered more or less random, though, it makes sense that LDA's Gaussian assumption will actually perform relatively well at finding a linear boundary between classes.

For readers interested in pursuing this research further: LDA might be interesting to look into for this task as more features are calculated and included in the feature vector. While most works we consulted seemed to rely on SVMs, these results imply that use of LDA in this setting shows some promise for future

research.

VI. Discussion and Future Work

The fields of computer forensics and computer security are quite large. There's a plethora of ongoing research in both of these areas, but despite all the attention it gets, machine learning has scarcely been applied to it. We found very few sources that cited the use of machine learning techniques for security and forensic applications, and those that did were usually in relation to anomaly-detection based Intrusion Detection Systems.

More common techniques for solving the fragment classification problem were systematic, and required an intimate knowledge of the format of each type of file, such as the header-footer method of file type detection described at the top. While this works for a small enough group of file formats, for general classification machine learning is surely more scalable, as hard-coding of headers and footers is unnecessary, and indeed impossible for recognizing some formats (JSON, etc.).

Our results are a promising sign that machine learning has a strong future in forensic and security applications, and can compete with less "intelligent" methods of classification and estimation, which is necessary if we wish to scale up to not just a few dozen, but hundreds or even thousands of file types.

There were a few issues with our dataset that may skew our results. The first is that the extensions of some files in the dataset are inaccurate, and while this does not give cause to disregard our results wholly, it does mean that there is some amount of error in our approximations. Considering the size of the dataset, fraction of mislabeled files has never been investigated, but after the author's individual probing, we found that most of the files we opened by hand matched their expected format, the most variance between actual and labeled format being between different plaintext files. For example, there were a few simple text files encountered that were labeled with a `.html` extension.

The other issue with the dataset is that in the end, the number of file types we train on does not even come close to the total nubmer of file types availabe "out in the wild". While it might not be feasible, or even useful, to try and build a classifier that recognizes all the thousands of file types, a mere 28 is not quite enough to cut it. While most research we consulted for this project had a similar or lesser number of file classes, eventually this work should be extended to include at least 100 common file formats for the tool to be at all practically useful.

For future research, the classification performed here is far from optimal, and lacks intelligence to distinguish well between multiple different types of the same format. For example, the classifier often misclassified `.html` files as `.py` files. This is in part due to the issues with the dataset as noted above, but in theory we can add a set of features for plaintext files which are calculated from syntax analysis of the text. This will allow us to eventually become even more precise in our classifications, as opposed to the accuracy that we were able to realize.

Eventually, this fragmentations process will not only be more accurate, but fully automatable and combinable with an algorithm to perform the fragment reassembly phase to achieve full carving capabilities.

References

[1] C. Beek, "*Introduction to File Carving*", McAfee Foundstone Professional Services, 2014.

[2] D. Ariu et al., *Machine Learning in computer forensics (and the lessons learned from machine learning in computer security)*, Proceedings of the 4th ACM workshop on Security and artificial intelligence, 2011.

[3] S. Fitzgerald et al., *Using NLP techniques for file fragment classification*, Digital Investigation, Vol. 9, 2012.

[4] S. Garfinkel et al., *Bringing science to digital forensics with standardized forensic corpora*, Digital Investigation, Vol. 6, 2009.

[5] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.