LOOKING FOR NON-COMPLIANT DOCUMENTS USING ERROR MESSAGES FROM MULTIPLE PARSERS

MICHAEL ROBINSON

ABSTRACT. Whether a file is accepted by a single parser is not a reliable indication of whether a file complies with its stated format. Bugs within both the parser and the format specification mean that a compliant file may fail to parse, or that a non-compliant file might be read without any apparent trouble. The latter situation presents a significant security risk, and should be avoided. This article suggests that a better way to assess format specification compliance is to examine the set of error messages produced by a set of parsers rather than a single parser. If both a sample of compliant files and a sample of non-compliant files are available, then we show how a statistical test based on a pseudo-likelihood ratio can be very effective at determining a file's compliance. Our method is format agnostic, and does not directly rely upon a formal specification of the format. Although this article focuses upon the case of the PDF format (ISO 32000-2), we make no attempt to use any specific details of the format. Furthermore, we show how principal components analysis can be useful for a format specification designer to assess the quality and structure of these samples of files and parsers. While these tests are absolutely rudimentary, it appears that their use to measure file format variability and to identify non-compliant files is both novel and surprisingly effective.

1. INTRODUCTION

Modern file formats are often quite complex, yet the formal specifications for some common formats can be ambiguous or confusing. A single parser is therefore not a reliable arbiter of file format compliance: it may incorrectly deem a compliant file as non-compliant, or conversely it may parse a non-compliant file (perhaps with disastrous consequences). For widely used file formats, there are usually several readily available parsers. It is natural to ask if a statistical approach that leverages multiple existing parsers – but is otherwise format agnostic – might suffice to discriminate between compliant and non-compliant files.

This article describes an exploratory technique and a statistical test for identifying files whose parser behavior is unusual. The techniques presented perform no direct inspection of the contents of any file. Certainly the content of a given file plays an important role in its usage, but the techniques of this article only "see the content" through the lens of an ensemble of parsers. Our techniques are therefore also well-suited to assessing the background variability of parser behavior on different classes of files. Since our approach aims to leverage existing parsers in their unmodified, uninstrumented state, the statistical techniques could be used on many different file formats without much alteration.

For the purpose of this article, a *file format* consists of a set of compliant files and a set of non-compliant files. Formal specifications specify properties that compliant files must have, but formal specifications need not be present for there to be an

agreed-upon file format. This article presents a new statistical test, which we call *the Bernoulli misclassification test*, that determines whether a given file is more representative of the compliant files or of the non-compliant files. In order to perform such a test, we require *samples* of both sets: namely a sample of compliant and a sample of non-compliant files. Because realistic samples of files are large and difficult to curate, the sample of compliant files may be contaminated with files that should not be considered as compliant. Conversely, there may be some files in our sample that are erroneously marked as non-compliant. Our statistical test is designed to identify these *misclassified* files.

The foundation of any statistical approach necessarily relies on both data coverage and sufficient sampling to ensure good estimation of the relevant governing parameters. Our approach here is no different, as the basis of the statistical test relies upon parameters estimated from the data in order to be effective. Given that our approach is format agnostic, it is reliant upon not only a good sample of files but also a good sample of parsers.

To test our approach, this article presents a case study using the PDF specification (ISO 32000-2), because there are many extant open source parsers with distinct underlying codebases. The test data presented in this article were developed by an independent test and evaluation team in support of the "DARPA SafeDocs Evaluation 2" exercise. The data consist of two datasets corresponding to the samples explained above: a sample of largely compliant files and a sample of largely non-compliant files. Each of the files (in both samples) was manually tested for compliance, so that the performance of our statistical test could be determined. While the fraction of misclassified files in the two datasets differ, the two datasets were sufficiently clean so as to allow reliable enough parameter estimation for our statistical test.

While the statistical methods discussed in this article are absolutely rudimentary, they did locate files that were truly misclassified with surprising effectiveness. Although these statistical methods surely do not suffice on their own for all purposes, they are easy to deploy and understand. We suggest that they ought to be part of the format hacker's toolbox.

2. Historical context

There appears to be very little work in analyzing file format compliance using statistical tools. In contrast to what we present here, most format compliance assessment that the author is aware of is performed using techniques common in compiler theory. For instance, [1; 11] explain the typical approaches.

The closest connections to this article appear to be various techniques for identifying malware using the structure of file *contents* rather than responses to those contents. For instance [6] looked for statistical features characteristic of malware present in headers of executable files. Using the structure of file contents, ransomware can be classified statistically [2; 12]. Similar to our use of error messages on files, the distribution of API calls can be useful in identifying malware as it executes [3]. Other behavioral indicators, such as performance counters [7] can be useful as well. However, it appears that the use of statistical tools to determine file format compliance is completely unanticipated and novel.

In a few cases, statistical methods are useful in identifying file formats that might be difficult to archive or curate [8; 10; 9].

TABLE 1. Counts of files in the `internet_sourced` and `dangerous` datasets

Dataset	Valid	Rejected	Total
`internet_sourced`	7206	1794	9000
`dangerous`	488	516	1000
Totals	7694	2306	10000

3. DATA DESCRIPTION

This article focuses on the analysis of PDF files, whose format is determined by the PDF specification (ISO 32000-2). It is recognized that the specification is ambiguous in places, and that there are many proprietary extensions to the specification. Because of this, many "PDF files" may not be completely compliant or may be close enough to compliance to parse correctly. On the other hand, bugs within the parsers may cause them to accept non-compliant files. To explore these issues, the test and evaluation team collected a corpus of "PDF files" into two datasets: `internet_sourced` and `dangerous` comprising a total of 10000 files.

Within each dataset, the files were manually determined to be either "valid", that is that they are compliant with the PDF specification, or "rejected", which means that they fail to comply with the specification. In this article, we treat the "valid" or "rejected" determinations as experimental *ground truth* for the files. Since our Bernoulli misclassification test did not use these determinations, we were able to use them to estimate the test's accuracy (discussed in Section 5). The overall statistics of both datasets are shown in Table 1. A standard χ^2 test reveals that the differences in compliance between the two datasets is statistically significant ($p < 0.0001$). Although it is far from true, we took the `internet_sourced` set to be our sample of predominantly compliant files, and we took the `dangerous` set to be our sample of predominantly non-compliant files. The significant difference between the two samples is precisely what our misclassification test leverages in order to find misclassified files.

Rather than looking at the contents of each file, we reasoned that there are already many extant parsers that do just that. Therefore, we ran each file through each of a large collection of parsers shown in Table 2. We selected these parsers based on their easy availability and with the understanding that many of them do not share code. This latter fact ensures that places within the PDF specification that are ambiguous may receive several interpretations by different parsers. The output to `stderr` from each parser was collected for each file, and a set of 955 regular expressions were used to identify which error messages had occurred for each parser-file pair (see Table 2).

As an example, the 1000-th file in the `internet_sourced` dataset was considered "valid," yet produced 7 distinct messages:

58: `caradoc_extract: Type error :  Invalid variant type`,
254: `caradoc_stats: Type error :  Invalid variant type`,
393: `caradoc_stats_strict: PDF error :  Syntax error`,
589: `hammer`: uncategorized error,
683: `pdfium`: uncategorized error,
910: `xpdf_pdfinfo`: uncategorized error, and
911: `xpdf_pdftops`: uncategorized error.

(a) internet_sourced   (b) dangerous
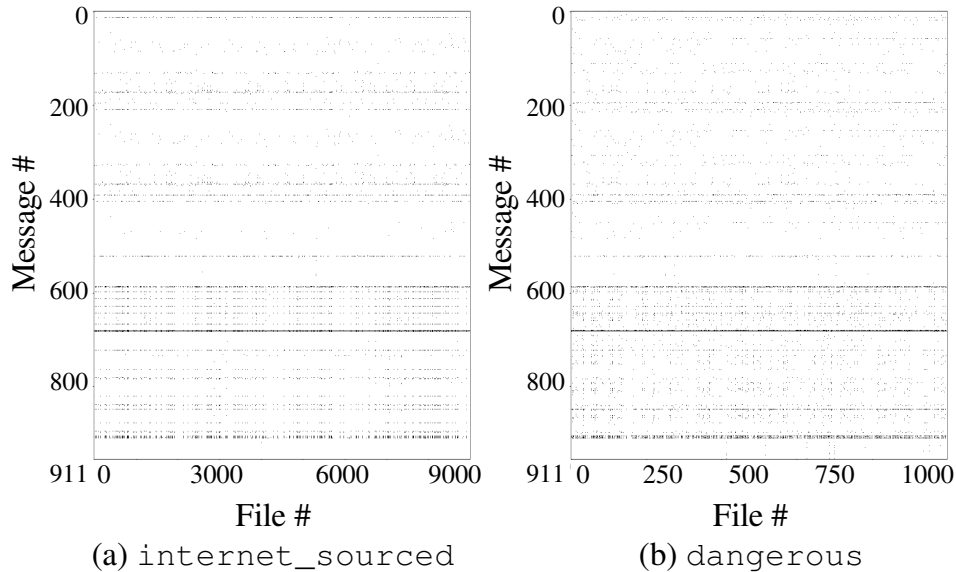
FIGURE 1. The relation matrix for (a) internet_sourced, (b) dangerous. Rows correspond to the error messages listed in Table 2. Columns correspond to files. In each matrix entry, white indicates no error, black indicates error.

It is important to recognize that our method *makes no attempt to interpret* the semantic meanings of these error messages. Instead, we are merely interested in the statistics of the co-occurrence of these messages.

The data can be tabulated as an integer *relation matrix*, in which each row corresponds to a particular regular expression (an *error message* in what follows) and each column corresponds to a particular file. Each entry records the number of times a given error message occurred for each file. Return values to the operating system were not collected, though if desired these could have simply been added as additional "messages" as rows in the relation matrix.

We constructed two relation matrices, one for internet_sourced (Figure 1(a)) and one for dangerous (Figure 1(b)). Each relation matrix has the same rows (955 distinct error messages, as shown in Table 2) but different numbers of columns (9000 for internet_sourced and 1000 for dangerous).

Continuing our example of the 1000-th file in internet_sourced, this file corresponds to the 1000-th column of the relation matrix in Figure 1(a), and has 1s in rows 58, 254, 393, 589, 683, 910 and 911, because each of these messages occurred exactly once. It has 0s in all other entries in that column.

The horizontal dark bands present in both relation matrices shown in Figure 1 correspond to error messages that could not be categorized easily: not syntax errors or other specific malformations. Some parsers produce these kind of messages more frequently than others, which explains why some portions of the matrices show a greater prevalence of dark horizontal bands than others.

Although there is some apparent structure visible in Figure 1, namely the dark horizontal bands, it is difficult to discriminate any column-by-column differences.

TABLE 2. Error message counts and rows per parser

Parser	First row	Last row	Total message regexes
caradoc_extract	1	196	196
caradoc_stats	197	392	196
caradoc_stats_strict	393	588	196
hammer	589	589	1
mutool_show	590	635	46
mutool_clean	636	681	46
origami_pdfcop	682	682	1
pdfium	683	683	1
pdfminer_dumppdf	684	703	20
pdfminer_pdf2txt	704	723	20
pdftk_server	724	724	1
pdftools_pdfid	725	729	5
pdftools_pdfparser	730	734	5
peepdf	735	735	1
poppler_pdfinfo	736	792	57
poppler_pdftocairo	793	849	57
poppler_pdftops	850	906	57
qpdf	907	907	1
verapdf_greenfield	908	908	1
verapdf_pdfbox	909	909	1
xpdf_pdfinfo	910	910	1
xpdf_pdftops	911	911	1
caradoc_extract	912	913	2
caradoc_stats	914	915	2
caradoc_stats_strict	916	917	2
hammer	918	919	2
mutool_clean	920	921	2
mutool_show	922	923	2
origami_pdfcop	924	925	2
pdfium	926	927	2
pdfminer_dumppdf	928	929	2
pdfminer_pdf2txt	930	931	2
pdftk_server	932	933	2
pdftools_pdfid	934	935	2
pdftools_pdfparser	936	937	2
peepdf	938	939	2
poppler_pdfinfo	940	941	2
poppler_pdftocairo	942	943	2
poppler_pdftops	944	945	2
qpdf	946	947	2
verapdf_greenfield	948	949	2
verapdf_pdfbox	950	951	2
xpdf_pdfinfo	952	953	2
Total			955

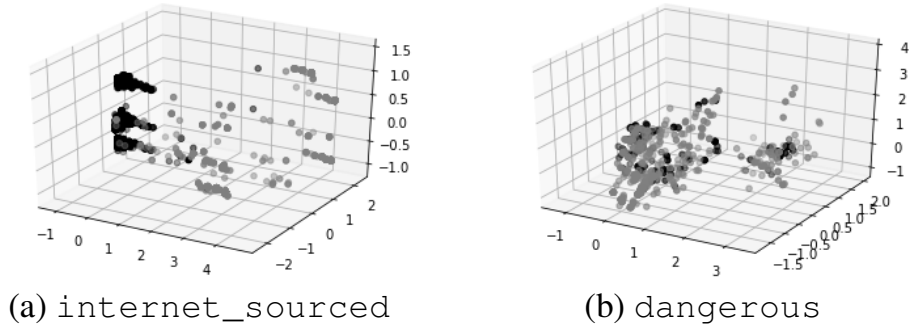(a) `internet_sourced`   (b) `dangerous`

FIGURE 2. Principal components plots for (a) `internet_sourced` and (b) `dangerous`. Black indicate "valid", and gray indicates "reject". The axes correspond to the three principal vectors, and so are not plotted on the same scale.

These differences are indeed present, but require more sophistication to extract. That statistical analysis forms the basis of most of this article.

4. PRINCIPAL COMPONENTS ANALYSIS

To build some intuition about the structure of the relation matrices, let us develop a dimension-reduced visualization of the columns (files) of both relation matrices shown in Figure 1. While there are many possible techniques for dimension reduction, principal components analysis is generally the easiest to construct and to interpret. To better understand the structure of the data, we will incorporate the ground truth as part of the visualization. This will help explain the performance of the Bernoulli misclassification test statistic in the next section.

Principal components analysis is a way to render a high dimensional data set that shows the "most important" dimensions and suppresses the rest. It is therefore a convenient way to visualize data that are formatted as a set of points in $\mathbb{R}^n$ where n is large. The output of principal components analysis is a scatter plot in which the axes are chosen as the linear combinations of rows yielding the largest variance. These axes are called the principal vectors, and are sorted from largest variance to least variance. In our analysis, the largest three principal vectors were used because they represented the majority of the variance in the data.

To apply principal components analysis, we reinterpret our tabular data as a discrete subset of $\mathbb{R}^n$ (a point cloud) in which columns (files) are points, rows (messages) are components of the coordinates for each point. In both datasets, there are $n = 955$ messages. Because a file exhibits an error or not, the components are all either 0 or 1. Although one might argue that this could result in quantization error, many interesting features are nevertheless visible in the two datasets.

Figure 2 shows the principal components analysis plots for both datasets. Points in both plots are colored according to the ground truth so that a point corresponding to a "valid" file is black, and a point corresponding to a "rejected" file is gray. The most striking difference in the principal components analysis plots is that the `internet_sourced` dataset is apparently much more "clumpy" than the `dangerous` dataset. The three dense clusters in Figure 2(a) consist entirely of "valid" files, with
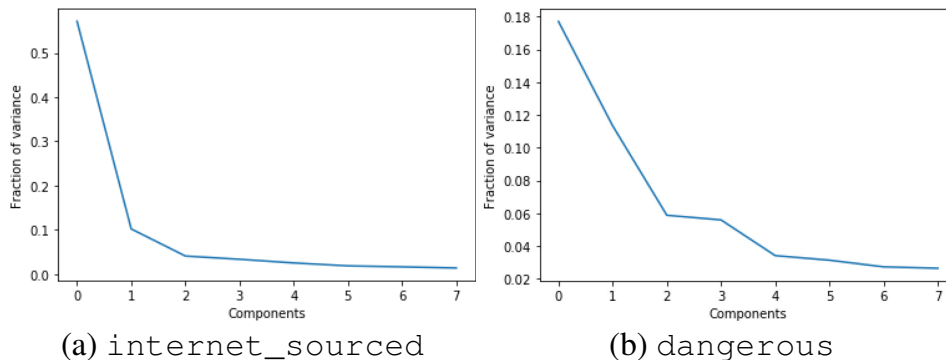
(a) `internet_sourced`   (b) `dangerous`

FIGURE 3. Scree plots for (a) `internet_sourced` and (b) `dangerous`.

most of the "rejected" files in the `internet_sourced` data appearing as a "haze" of files outside of those clusters. Although the cause of these three dense clusters of "valid" files cannot be determined solely from the relation matrix – which files are accepted by which parsers – we hypothesize that these clusters correspond to popular tool chains for creating PDF files.

In stark contrast, the `dangerous` set shown in Figure 2(b) contains two loose clusters that are mixed "valid" and "rejected" files. Intuitively, if one is looking to identify "valid" files, one would have a much harder time doing this with the `dangerous` set, so we might argue that the apparent signal-to-noise ratio is much lower in the `dangerous` set.

Principal components analysis can be misleading if only a small fraction of the overall variance in the data is explained by the first few principal vectors. It is easy to determine if this problem is occurring – simply plot the variance in the data explained by each principal vector. This is called a Scree plot [13], and is shown in Figure 3. In both datasets, the Scree plots decrease quite rapidly over the first few principal vectors. This shows that principal components analysis reliably represents the data.

We can conclude that if one is generally working with files that are naturally occurring (like the `internet_sourced` set), one probably does not need to dig too deeply to determine whether a given file is valid or not. The rest of this article buttresses this claim by providing a statistical test that does just that. On the other hand, if one is routinely handling files that test the limits of their format (like the `dangerous` set), statistical analysis alone will likely be insufficient to determine which files are valid. A deeper format-aware analysis would be necessary in that case.

5. BERNOULLI LIKELIHOOD RATIO MISCLASSIFICATION TEST STATISTIC

In order to determine file validity implicitly – by consulting parser behavior rather than the specification itself – we need exemplars of parser behavior on typical compliant and non-compliant files. We propose to use the fact that the two datasets have rather different statistics (Table 1) with the majority of files in `internet_sourced` being "valid" while a (slim) majority of files in `dangerous` are "rejected". We can attempt to identify files in `internet_sourced` that exemplify

the parser behavior present in the `dangerous` dataset – these files probably should be treated with suspicion, and are perhaps not "valid." Conversely, files within the `dangerous` set that behave more like those in `internet_sourced` may well be "valid."

A file is *misclassified* if the set of messages it produces is more characteristic of the other dataset but not the one in which it is presently found. We suspect that such a misclassified file will have a collection of error messages that is unusual when compared to the others in that dataset. We estimate the probability that each error message occurs in each dataset, and estimate a likelihood for each file assuming the error messages are independent. Since both datasets have the same error messages, we can *also* estimate the likelihood that the file came from the other dataset. A likelihood ratio statistic can thereby detect when a file is misclassified, because it is more likely to belong to the other dataset.

Since we cannot assume that the occurrence of any given set of error messages is statistically independent, it is difficult to write a proper likelihood function. To that end, we use a *pseudo-likelihood*, which makes the incorrect assumption that error messages are statistically independent [5]. On the other hand, this assumption merely reduces the sensitivity of the test without producing extra outliers. The pseudo-likelihood assumption trades *recall* to get better *precision*. As such, pseudo-likelihoods are useful for classification but not for uncertainty quantification.

For the Bernoulli misclassification test, we assume each error message is governed by a Bernoulli distribution, which means that it either occurs or does not occur. Multiple instances of the same error are ignored. The test assumes that each error message occurs with a probability that depends on the dataset (either `internet_sourced` or `dangerous`). When a parser processes a given file, sometimes it produces multiple copies of the same error message. This can happen if the parser attempts to repair a slightly non-compliant file as it proceeds, for instance. If this happens, then the Bernoulli distribution is no longer valid because it assumes at most one instance of a given error message. Ultimately, the performance of the Bernoulli misclassification test was good even though we ignored duplicate error messages.

Let us consider the `internet_sourced` dataset first. It is straightforward to compute the probability p_k of error message k occurring from the relation matrix: simply take the average value of row k in the relation matrix shown in Figure 1(a). The resulting probabilities for both datasets are shown in Figure 4. Said another way, if file f is drawn from the `internet_sourced` dataset, then the probability that f produces error message k is p_k. Conversely, the probability that f does *not* produce this error message is $(1 - p_k)$. If we let $f_k = 0$ if the file f did not produce error k and $f_k = 1$ if the file did produce error k, then the probability that f is from the `internet_sourced` dataset is

$$p_k f_k + (1 - p_k)(1 - f_k),$$

if we only consider error message k.

Since we have many error messages available for analysis, the pseudo-likelihood that file f (column in the relation matrix) is correctly classified is simply the product of each of these individual probabilities, namely

$$L_{\texttt{internet\_sourced}}(f) = \prod_{k=1}^{955} (p_k f_k + (1 - p_k)(1 - f_k)).$$

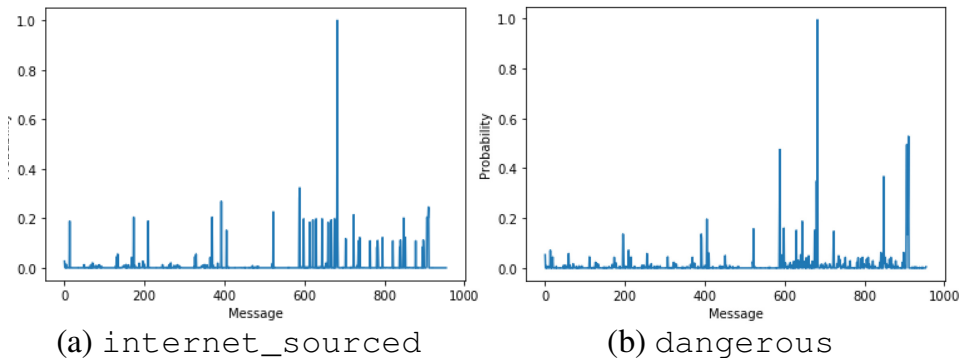**(a)** `internet_sourced`   **(b)** `dangerous`

FIGURE 4. Error probability for (a) `internet_source` and (b) `dangerous`.

We define $L_{\texttt{dangerous}}(f)$ similarly using the error message probabilities p'_k from the `dangerous` set instead,

$$L_{\texttt{dangerous}}(f) = \prod_{k=1}^{955}(p'_k f_k + (1 - p'_k)(1 - f_k)).$$

We define the *Bernoulli misclassification test statistic* to be the ratio of these two pseudo-likelihoods,

$$\lambda_{\texttt{internet\_sourced}}(f) = \frac{L_{\texttt{dangerous}}(f)}{L_{\texttt{internet\_sourced}}(f)}.$$

If f is a file drawn from the `internet_sourced` dataset, then we generally expect that $L_{\texttt{internet\_sourced}}(f)$ will be larger than $L_{\texttt{dangerous}}(f)$, which implies that $\lambda_{\texttt{internet\_sourced}}(f) < 1$. Conversely, if a file is drawn from the `dangerous` dataset, which means it is a misclassification if it actually is present in `internet_sourced`, we would expect that $\lambda_{\texttt{internet\_sourced}}(f) > 1$. The intuition is that since files in the `dangerous` set are likely to be invalid, a high value of $\lambda_{\texttt{internet\_sourced}}(f)$ is an indication that the file f is not compliant. A histogram of $\lambda_{\texttt{internet\_sourced}}$ is shown in Figure 5(a).

Conversely, we can define

$$\lambda_{\texttt{dangerous}}(g) = \frac{L_{\texttt{internet\_sourced}}(g)}{L_{\texttt{dangerous}}(g)}$$

for each file g in the `dangerous` set. The intuition in this case is that a high value of $\lambda_{\texttt{dangerous}}(g)$ is an indication that g is compliant, since it is likely to be a misclassification. The histogram of values of $\lambda_{\texttt{dangerous}}(g)$ is shown in Figure 5(b).

Since there is some variability (or noise) present within the data, we should not use the value of $\lambda = 1$ as the cutoff for detecting misclassifications. Although the intersections between the histogram curves and the red lines $\lambda = 1$ in both frames of Figure 5 are close to the true fraction of misclassifications in both datasets (80% for `internet_sourced` and 48% for `dangerous`), they are not exactly correct. This suggests using a different threshold T instead, so that all files whose statistic λ is greater than T will be deemed to be misclassified.

Let us now use the ground truth, which specifies whether a given file is "valid" or "rejected", to determine the performance of the Bernoulli misclassification test
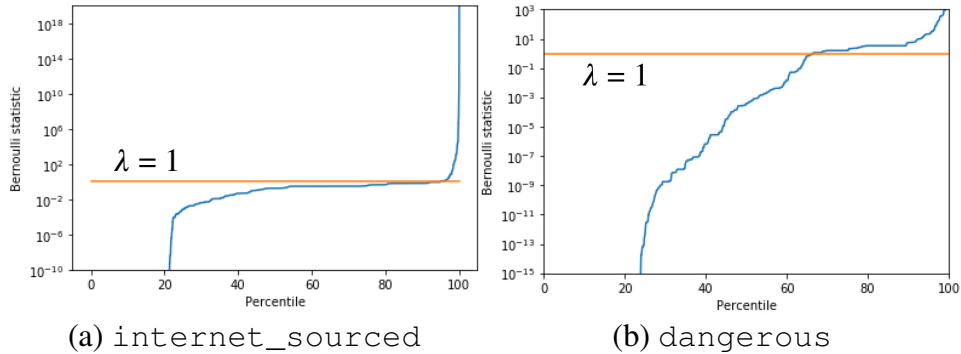
(a) internet_sourced   (b) dangerous

FIGURE 5. Histogram of Bernoulli misclassification test statistic
for (a) internet_sourced and (b) dangerous. The horizontal line
marks a value of $\lambda = 1$.
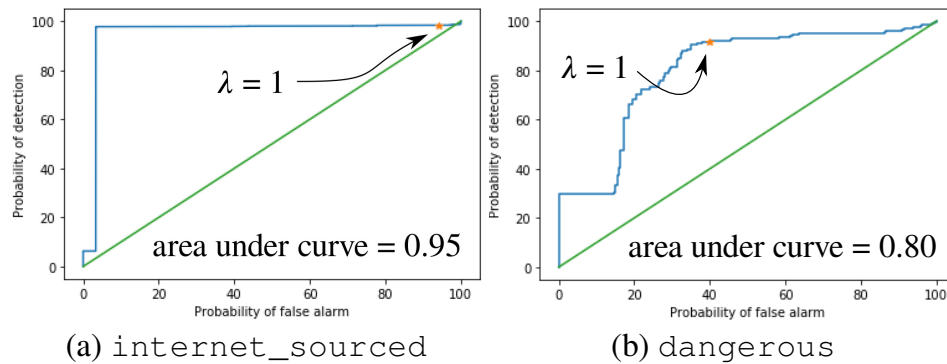

(a) internet_sourced   (b) dangerous

FIGURE 6. Receiver operating curves for Bernoulli likelihood ratio
misclassification statistic for (a) internet_sourced and (b)
dangerous.

statistic. A misclassified file in internet_sourced is one that is "rejected", while
a misclassified file in dangerous is one that is "valid".

In the case of either dataset, for a given threshold T, the *probability of detection*
is the probability that a truly misclassified file f will have a statistic $\lambda(f) > T$.
On the other hand, the *probability of false alarm* is the probability that a correctly
classified file f will have a statistic $\lambda(f) > T$. Ideally, the probability of detection
will be close to 1, while simultaneously the probability of false alarm will be close
to 0.

The plot of probability of detection versus false alarm for all thresholds is shown
in Figure 6. Better misclassification detectors have plots further to the upper
left, away from the diagonal. Since the curve plotted is far above the diagonal
for internet_sourced in Figure 6(a), we conclude that the Bernoulli likelihood
ratio misclassification statistic is a very accurate detector of misclassified files in
internet_sourced. Additionally, since the plot in Figure 6(b) is above the diago-
nal, this indicates that the Bernoulli likelihood ratio misclassification statistic also

performs well on the `dangerous` dataset set. The sharp plateau in Figure 6(b) is due to a number of instances in which $L_{\mathtt{internet\_sourced}}$ took the value 0 on some files in `dangerous`. These happen to be truly non-compliant files! These qualitative assessments are confirmed by the areas under the curves in Figure 6. A perfect misclassification detector will have an area of 1 under the curve, while a detector that randomly decides misclassifications would have area 0.5 under the curve. We obtained 0.95 for $\lambda_{\mathtt{internet\_sourced}}$ and 0.80 for $\lambda_{\mathtt{dangerous}}$ for areas under the curve, which we deem to be surprisingly good given the fact that the file contents were not directly inspected by our method.

6. Parser redundancy analysis

The Bernoulli misclassification statistic relies upon the diversity of responses to each file in order to make reliable decisions. It is impractical to use quite so many parsers as we used, so it is useful to assess whether we could get good performance with fewer parsers. The easiest way to do this is to measure the correlation between the typical error message distributions produced by different parsers across all files. In the following analysis, we combined both `internet_sourced` and `dangerous` into a single dataset, whose relation matrix consists of the horizontal concatenation of both matrices shown in Figure 1. That is, the rows are the same as before, but the columns consist of the union of the columns from both matrices. Error messages (rows) that never occur were removed from the analysis, since they contribute no information.

The data can also be visualized using principal components analysis, much as was done in Section 4, but instead we use the error counts across files as coordinates for each parser. Since specific error messages cannot be enabled or disabled individually, it makes sense to aggregate the error messages into parsers by taking the total number of error messages for a each parser on a given file. Principal components analysis places the parsers according to the plot shown in Figure 7. There is one large cluster (in the lower left of Figure 7) of parsers which have similar behaviors. There are quite a few outliers, most notably `caradoc_extract` and `poppler_pdfinfo`. The reader is cautioned that the apparent density of the large cluster is a bit misleading, because the ranges of the axes are quite different. In any event, the wide spread of parsers across the plot indicates that having a large number of parsers is quite valuable.

Although principal components analysis is useful, we can obtain a ranking of parsers by their redundancy, so that we can prioritize more informative parsers. We computed Pearson's correlation coefficient between all pairs of error messages (rows), to form a fairly large correlation matrix (not shown). Different error messages that occur on exactly the same set of files have a correlation coefficient of 1. In that case, one of those error messages is redundant. Errors were then grouped by parser, and for each pair of parsers we stored the median correlation from each of their pairwise message correlations. The result of this aggregation is the matrix shown in Figure 8. Whiter colors indicate higher correlation – more redundancy – while darker colors indicate lower correlation. Most of the matrix is fairly dark, which indicates low redundancy overall. The bright off-diagonal entries indicate trade-offs: one needs only run one of the two parsers indicated. The bright bands for `pdftools_pdfid` occurred because that parser did not correlate with any of the others at all.

FIGURE 7. Parsers placed in three dimensions according to principal components analysis of the union of `internet_sourced` and `dangerous`.

The clusters visible in the Figure 7 can be confirmed by sorting the parsers based on their median correlation. The rows and columns shown in Figure 8 are sorted in this way. This indicates which parsers are individually most informative, because they form an approximate spanning set for the data. The obvious block in the lower right suggests that parsers should be grouped into two categories based on their informativeness: high and low. The rows and columns of the block in Figure 8 suggest that the boundary between the high and low categories appears to be fairly sharp, with all parsers to the left of `origami_pdfcop` being highly informative.

Based on the approximate spanning property of the highly informative category, which covers most of the variability visible in Figure 7, one should always run the parsers in the high category, with the other parsers in the low informative category parser treated as increasingly optional as one moves to the lower right of Figure 8. Notice that the parsers near the lower right of Figure 8 all come from the cluster
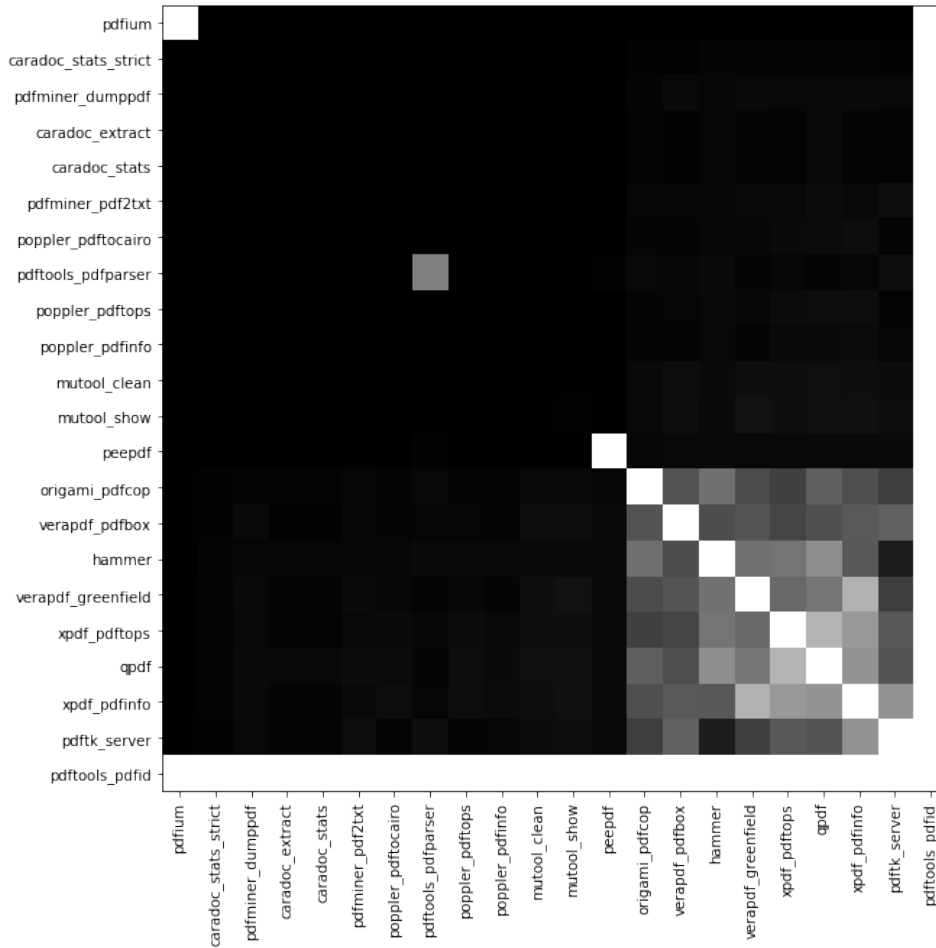
FIGURE 8. Correlation matrix between parser behaviors. Rows
and columns are sorted according to median message correlation.

in the lower left of Figure 7. Although it is difficult to see from the projection
shown in Figure 7, least informative parsers are on the *inside* of the cluster, while
the parsers in the same cluster that are on the outer edges of the cluster are more
informative.

7. CONCLUSION AND RECOMMENDATIONS

This article has demonstrated that a given file's compliance with a format speci-
fication can be determined from the following three samples: (1) a diverse collection
of parsers for the file format, (2) a sample of compliant files, and (3) a sample of
non-compliant files. Our methodology is format agnostic, and so could work if a
file format is not formally specified. Furthermore, our Bernoulli test for compli-
ance is statistical, so it is both robust to errors in the three samples, yet benefits
from richer samples should they be available. We note that the use of the three
samples means that this approach is a supervised approach. Under appropriate

circumstances, it might be possible to use an unsupervised bootstrapping approach to extract the two samples of files from a single aggregated sample. This requires further investigation.

Principal components analysis is helpful in understanding the variability within a sample of files or parsers, but it holds somewhat less value as an automated analytic tool. Clusters visible within the principal components plots appear to reflect specific tool chains used in the creation of files, with valid files forming several dense clusters.

The Bernoulli likelihood ratio statistic detects non-compliant files by comparing error message prevalence aggregated across all parsers and both samples of files. It relies inherently upon both the coverage and depth of these samples, but we have shown that it can be very effective at its job when these samples are adequate.

Overall, we found that there is not much redundancy present in the behaviors of parsers for the PDF specification. That is to say that the relation matrix contains by far the most information if all parsers are considered, so one ought to use all parsers if possible. Our analysis determined which parsers are individually the most informative, based on pairwise comparisons. While it is entirely reasonable to study different subsets of parsers, we did not perform that analysis here. We refer the interested reader to [4] where such an analysis was performed.

If resources are tight, we found that it is probably not necessary to rerun a given parser with different options. For instance, running only one of `xpdf_tops` and `xpdf_pdfinfo` probably will not change the results too much. Roughly speaking, it appears that the best strategy is "more programmers contributing different code instead of one programmer's code run in many different ways."

Our two statistical techniques for analyzing file format compliance are admittedly simple but standard statistical tools, though are apparently not in wide use. They are easy to deploy, easy to interpret, and require little maintenance other than the selection of a single detection threshold. We therefore encourage deeper exploration into and experimentation with statistical methods in the study of file format compliance.

References

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. Compilers, principles, techniques. *Addison wesley*, 7(8):9, 1986.

[2] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security*, 74:144–166, 2018.

[3] Mamoun Alazab. Profiling and classifying the behavior of malicious codes. *Journal of Systems and Software*, 100:91 – 102, 2015. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2014.10.031. URL `http://www.sciencedirect.com/science/article/pii/S0164121214002283`.

[4] Kristopher Ambrose, Steve Huntsman, Michael Robinson, and Matvey Yutin. Topological differential testing. *arXiv preprint arXiv:2003.00976*, 2020.

[5] Barry C Arnold and David Strauss. Pseudolikelihood estimation: some examples. *Sankhyā: The Indian Journal of Statistics, Series B*, pages 233–243, 1991.

[6] Mohamed Belaoued and Smaine Mazouzi. A real-time PE-malware detection system based on chi-square test and PE-file features. In *IFIP International Conference on Computer Science and its Applications*, pages 416–425. Springer, 2015.

[7] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[8] Roman Graf and Sergiu Gordea. A risk analysis of file formats for preservation planning. In *Proceedings of the 10th International Conference on Preservation of Digital Objects (iPres2013)*, pages 177–186, 2013.

[9] Gregory W Lawrence, William R Kehoe, Oya Y Rieger, William H Walters, and Anne R Kenney. *Risk Management of Digital Information: A File Format Investigation.* ERIC, 2000.

[10] D. Pearson and C.Webb. Defining file format obsolescence: A risky journey. *The International Journal of Digital Curation*, 3(1):89–106, July 2008.

[11] Maksym Schipka. Detection of exploits in files, January 8 2009. US Patent App. 11/822,533.

[12] S. D. S.L and J. CD. Windows malware detector using convolutional neural network based on visualization images. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2019. doi: 10.1109/TETC.2019.2910086.

[13] An Gie Yong, Sean Pearce, et al. A beginner's guide to factor analysis: Focusing on exploratory factor analysis. *Tutorials in quantitative methods for psychology*, 9(2):79–94, 2013.

MATHEMATICS AND STATISTICS, AMERICAN UNIVERSITY, WASHINGTON, DC, USA
Email address: `michaelr@american.edu`