Malicious HTML File Prediction: A Detection and Classification Perspective with Noisy Data

Samuel Hess, Pratik Satam, Gregory Ditzler and Salim Hariri

Abstract— **Cybersecurity plays a critical role in protecting sensitive information and the structural integrity of networked systems. As networked systems continue to expand in numbers as well as in complexity, so too does the threat of malicious activity and the necessity for advanced cybersecurity solutions. Furthermore, both the quantity and quality of available data on malicious content as well as the fact that malicious activity continuously evolves makes automated protection systems for this type of environment particularly challenging. Not only is the data quality a concern, but the volume of the data can be quite small for some of the classes. This creates a class imbalance in the data used to train a classifier; however, many classifiers are not well equipped to deal with class imbalance. One such example is detecting malicious HMTL files from static features. Unfortunately, collecting malicious HMTL files is extremely difficult and can be quite noisy from HTML files being mislabeled. This paper evaluates a specific application that is afflicted by these modern cybersecurity challenges: detection of malicious HTML files. Our previous work presented a general framework for malicious HTML file classification that we modify in this work to use a χ^2 feature selection technique and synthetic minority oversampling technique (SMOTE). We experiment with different classifiers (i.e., AdaBoost, GentleBoost, RobustBoost, RusBoost, and Random Forest) and a pure detection model (i.e., Isolation Forest). We benchmark the different classifiers using SMOTE on a real dataset that contains a limited number of malicious files (40) with respect to the normal files (7,263). It was found that the modified framework performed better than the previous framework's results. However, additional evidence was found to imply that algorithms which train on both the normal and malicious samples are likely overtraining to the malicious distribution. We demonstrate the likely overtraining by determining that a subset of the malicious files, while suspicious, did not come from a malicious source.**

I. Introduction

The significant growth of communication systems has created an environment where many of us rely on networked systems in our daily life. Examples include online banking systems that use pin and social security numbers or a national government defense network. Each of these applications are commonly used in many of our daily lives; however, they contain sensitive information that if compromised could have a devastating impact. If not appropriately protected, this delicate information can be obtained and exploited by a malicious user [1]. Additionally, these networked systems have grown exponentially over the past decade and continue to grow. This growth has naturally created an increased demand for automated cybersecurity solutions [2], [3]. Traditionally intrusion detection systems have been signature based[4]. The use of signature based systems is not effective as it has been observed that they have a very high false negative rate, making the users doubt their reliability. In the recent time, there is a shift in trend towards modeling based intrusion detection systems that use mathematical models. These mathematical models are an improved representation of the target system compared to the signature based approach and have been seen to have lower false negative rate. Moreover as the performance characteristics of the hardware on which the intrusion detection systems run have improved, these systems have moved towards having more automated responses to the detected threats[5]

Cybersecurity remains a challenging task for modern automated systems because: (i) many supervised machine learning algorithms rely on a significant amount of labeled data with limited noise; (ii) the number of malicious data generated by the adversary is quite small in comparison to the total number of data from normal activities; (iii) the adversary has the ability to develop new attack strategies that were not available at the time of training the machine learning model. For this work, the adversary's goal is to create a malicious file to collect information from a system or cause internal harm. Furthermore, the adversary's intention is to be able to achieve these objectives before being detected. New cybersecurity methods are needed to overcome older security systems if the malicious file is quickly exposed. Many of the current automated cybersecurity solutions make use of prior knowledge of malicious behaviors to identify new threats. As a result, an evolving threat makes it difficult for the automated protection system to provide reliable predictions if it was trained on old data. Additionally, it is easier to establish data on normal files than malicious files due simply to the fact that the number of occurrences from normal activity generally far exceed the number of samples from malicious activity. Thus, once data is collected, the system contains lot of knowledge about the distribution of "normal" files whereas very little knowledge about the distribution of the emerging malicious threats. This unequal ratio of data available to the system for training is known as data imbalance and is known to create difficulties in classification algorithms.

This paper addresses the application of cybersecurity for malicious HTML detection with static features that are collected from an imbalanced dataset. Specifically, we look at a framework for detection that includes extracting features from HTML files, preprocessing steps (i.e., a sampling

S. Hess, P. Satam, G. Ditzler and S. Hariri are with the Department of Electrical & Computer Engineering at the University of Arizona, Tucson, AZ 85721. P. Satam and S. Hariri are affiliated with the NSF Center on Cloud and Autonomic Computing.

{shess,pratiksatam,ditzler,hariri}@email.arizona.edu

technique [6] and χ^2 feature selection [7]), and using detection and/or classification algorithms (i.e., Isolation Forest [8], CART [9], Random Forest [10], Adaboost.M1 [11], GentleBoost [12], RUSBoost [13], and RobustBoost [14]) on those processed features to obtain a final prediction for an HTML file. Additionally, we use a database of HTML files that is significantly imbalanced (1:180 imbalance ratio of malicious to normal HTML files). We assess the performance of the various configurations of this framework using a cross-validation method on our collected database of 7,303 samples.

Using the established framework, the system performance obtained on our dataset has an accuracy of 99.99% with a true positive rate (sensitivity) of 100% and a true negative rate (specificity) of 97.5%. This performance was achieved by using SMOTE of 1000% with the GentleBoost classification algorithm. Additional analysis on our dataset empirically show that many of the proposed algorithms have over fitted to the minority distribution in our dataset. However, by using feature selection we've demonstrated that we can reduce the effect of over fitting and improve robustness at the slight cost of a degradation in system accuracy and sensitivity.

II. Related Work

The amount of research related to cybersecurity and machine learning is rather extensive and covers an enormous variety of applications. This application of malicious HTML file detection is often categorized as static or dynamic detection. Static detection uses features extracted directly from the HTML file to detect, whereas dynamic detection creates features from the execution of the HTML. Dynamic analysis often is performed in a protected sandbox environment to prevent harm of executing the file on an actual system, which tends to be computationally expensive. In this paper, we specifically focus on static methods to extract features from an HTML file and apply machine learning techniques to detect malicious HTML files. For this reason, the related work is organized into two sections. The first section reviews common static properties of HTML files that creators of malicious content often exploit. These properties are hidden Iframes, malicious references, malicious scripts, and abnormal construction. The second section reviews existing detection frameworks and techniques that have applied support vector machines, decision trees, and ensemble methods to static HTML detection.

A. Attacking via HTML files

HTML is a highly-structured language, and there are four techniques for transforming legitimate HTML files into a malicious file, which are discussed below.

1) Hidden Iframes: A webpage can load another page inside itself using an Iframe or "Inline Frame". The webpage that is loaded using an Iframe is not restricted to the same server. Iframes are useful while creating online applications or hosting content (such as videos), which are located on a different source. It is possible to set the size of an Iframe to 0 pixels, and/or give them coordinates such that the frame is

```
<iframesrc="http://www.MaliciousWebsite.com" width="1"
height="1"></iframe>
```

Fig. 1. Example of a malicious IFrame that could appear in an HTML file.

```
<div class="views-field views-field-body">
<divclass="field-content"><p>

<iframe height="0"
src=http://103.27.108.45/img/js.php width="0">
        </iframe></p>
```

Fig. 2. An Iframe of size zero

off the page and/or set the property of the Iframe to "hidden", thus effectively rendering the Iframe invisible. Such invisible Iframe can be used to run malicious Java scripts or remote malicious page downloading codes. Figure 1, Figure 2 and Figure 3 show Iframes with small size loading a link to a malicious webpage and a malicious code.

2) Malicious Reference: A benign webpage can easily be converted into a malicious webpage by linking it to a malicious webpage. The benign webpage generally has an hyperlink that links it to a malicious webpage or a malicious file. Such malicious hyperlinks can be easily inserted into an HTML file using the "meta" tag inside the HTML head, or the "a" tag inside the HTML body. As shown in the figure below, the "a" tag can be used in the HTML body to create a hyperlink to another malicious webpage or an hyperlink to download a malicious file. Figure 4 Figure 5 and Figure 6 show some implementations of the malicious references.

```
<iframesrc="http://www.
    MaliciousWebsite.com"width="1"
    height="1"></iframe>


<divclass="views-field␣
    views-field-body">
<divclass="field-content"><p>

<iframe height="0"
src=http://103.27.108.45/img/js.php
    width="0">
</iframe></p>
```

Fig. 3. An Iframe with a malicious reference

```
<meta http-equiv="location"
content="url=http://www.MaliciousSite.com" />
```

Fig. 4. A meta information to a malicious page.

```
<ahref="http://www.maliciousSite.com"?>Wonderful
Website</a>
```

Fig. 5. A hyperlink to a malicious website.

3) Malicious Scripts: JavaScripts are embedded into webpages to make them dynamic. However, JavaScripts can be easily used to perform malicious tasks. JavaScripts have features that allow runtime creation of code, and runtime execution of text through D-Gen and R-Eval respectively. These features when combined with the inbuilt functions like unescape and `string.fromCharCode`, allow a high level of code obfuscation, making it even harder to detect malicious Javascript in a webpage. There are almost unlimited ways to use JavaScript maliciously, and a few of them are in the following list:

1) Read files from local drive
2) Fill up a local drive
3) Access or replace files on the local machine
4) Close or open windows
5) Launch an application
6) If the browser allows, read browser history or cookies
7) Exploit bugs in a browser
8) If the browser allows, load another document or script from a different domain

4) Abnormal Construction: This category contains several different malicious activities such as: seldom used tag names, HTML structure inconsistencies, and malicious obfuscation. Such abnormal construction features are generally combined with the D-Gen functions of Javascripts to create runtime malicious behavior.

B. HTML Attack Detection Methods

In [15] the authors conducted a survey of techniques to obfuscate malicious Javascript. Their work describes the different types of approaches taken to make a Javascript malicious. They also highlight subtle differences between benign but obfuscated Javascript and malicious and obfuscated Javascript[c1]. In [16], a feature based approach was presented by the authors to detect malicious HTML files using a two phase detection method. In the first stage, an anomaly behavior analysis approach uses a C4.5 decision tree algorithm to classify a webpage to be malicious. In the second stage, a detection module uses a one-class support vector machine to reduce the errors from the files classified as malicious in the stage 1. In [17] the authors used a feature based approach similar to our approach to detect malicious HTML files. They used 14 feature set to characterize the HTML compared. In [18], the authors present a half dynamic classification method to detect malicious Javascript. They use an *n*-gram based approach to perform a half-dynamic analysis of the HTML file. Another recent work presented a dynamic analysis technique that measures various factors like the average script execution time, maximum execution time, number of function calls, and total URL sets in the file to classify the files using different machine learning models [19].

```
<ahref=http://213.171.193.5?20/MaliciousAttk/at
tack.php download="UsefulFile">
```

Fig. 6. Downloads malicious file on the local machine.

In Eshete et al. [20] the authors use 25 features as a part of page source features to classify normal pages and malicious pages. This set of 25 features is obtained from previous work and is a subset of the dataset that we have used as a part of our work. Moreover, the authors use features pertaining to URL of the webpage and social-reputation of the page to improve the performance of their IDS. As a part of our work we have restricted ourself to the use of features pertaining to webpages and extended our work to include URLs and social reputation, because our IDS is aimed at the user trying to ensuring that his webpages are clean, implying that URL based features are not required. Moreover, we feel that social reputation of a page depends on the popularity of the page and thus a large percentage of the webpages on the internet will have little or no reputation on the social media.

III. Methodology for Malicious HTML Detection and Validation

This paper extends on the material presented by Satam et.al [21]. In our previous work, we presented an IDS that used two layers of classification models to classify malicious HTML files. They used a set of 32 features to characterize the HTML files and the feature selection algorithm presented by Qu et al. [22] to obtain a reduced feature set. Additionally, they used a two layer classification system to classify the malicious files. The first layer involved the use of C4.5 decision tree classifier, regression classifier, and bagging classifier whose results were combined in the second layer with the use of a C4.5 classifier. We obtained a true positive rate (TPR) of 99% and a false positive rate (FPR) of 0.8%. The work presented here follows a similar methodology using the same data collection and feature extraction process. As a part of this work, we attempted to address the data imbalance by using synthetic minority oversampling (SMOTE) [6]. Furthermore, the detection and/or classification algorithms

[c1] *greg*: Huh?

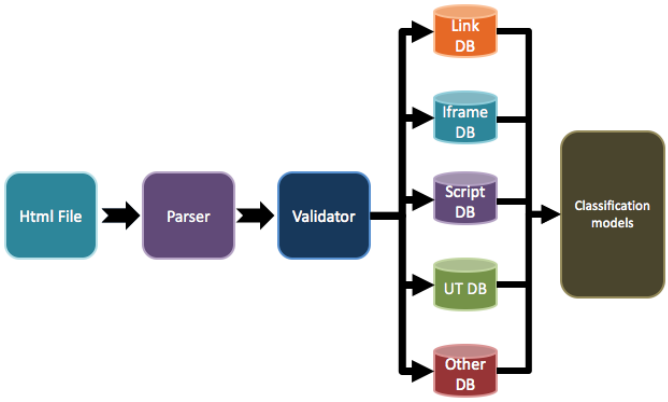

Fig. 7. DAE approach to analyze HTML files.

have been modified to use ensemble boosting and bagging algorithms such as Adaboost, Random Forest, and Isolation Forest.

The described framework for malicious HTML detection is known as the Data Analysis Engine (DAE) module and is illustrated in Figure 7. The following subsections describe the various sub-modules of the DAE as well as the details of training and testing the classification models.

A. Data Collection and Feature Extraction

The different sub-modules that constitute the DAE shown in Figure 7 are:

1) *Parser*: As shown in Figure 7, the parser is the first block to receive the HTML file from the user. The parser's job is to parse the HTML file then ensure that the HTML file is structurally intact and extract the features from the HTML file. Moreover, the parser marks an HTML file malicious if it has a malicious link which is matched from the malicious link database. The complete feature set is shown in Table I. The highlighted entries in the table are the features that remain after χ^2 feature selection is applied. One of the motivations for feature selection in this setting is to prefer similar models, which can be beneficial for a detector (e.g., Isolation Forest) since the model is only trained on one class.
2) *Validator*: The validator checks if the data that is extracted by the extractor is correct. This module ensures that the data is in valid boundaries and format.
3) *Classifier*: The classifiers are machine learning algorithms that are obtained as a part of the training phase of the system. The classifiers take in the features values that are extracted by the validator's to classify if a file has normal behavior. The classification model consists of three distinct phases: a training phase, an validation phase, and an operational phase.

The training and validation phase are shown in Figure 8 and is presented in more detail in the following section.

TABLE I
LIST OF COLLECTED STATIC HTML FEATURES. HIGHLIGHTED FEATURES ARE SUBSET SELECTED BY χ^2 FEATURE SELECTION

Collected Feature Set	
1) total forms	17) external iframe
2) external forms	18) total objects
3) total links	19) total characters
4) external links	20) keyword count
5) max length of links	21) keywords to words ratio
6) min length of links	22) white space ratio
7) average length of links	23) script length
8) unique tags	24) string modification
9) total tags	25) dom modification
10) total scripts	26) interaction events
11) external scripts	27) event attachments
12) obfuscated html	28) total redirects
13) native functions	29) string length
14) set timeout	30) max entropy
15) total iframes	31) min entropy
16) hidden iframes	32) total entropy

B. Training and Evaluating the Classification models

After the database of features have been collected from the HTML files, it is the objective to establish a model from this information that can be used to accurately detect malicious HTML files. To achieve this task, there are a number of detection and/or classification algorithms available in the literature. Strictly speaking, detection algorithms attempt to determine when a given sample is outside the expected normal model whereas classification algorithms attempt to find the best model to categorize a given sample. In this work, we are currently only concerned with detecting the malicious files. However, if there is sufficient information collected on both classes of data (i.e. normal and malicious), it is possible that a two-class classification algorithm would use more of the collected information and increase detection sensitivity. For this reason, our detection framework has experimented with both detection algorithms, such as one-class support vector machines (one-class SVMs) [23] and Isolation Forest [8], as well as classification algorithms, such as a standard classification tree [9], Random Forest [10], Adaboost.M1 [11], and variants of Adaboost (i.e. GentleBoost [12], RUSBoost [13], and RobustBoost[14]).

One disadvantage of using classification techniques for malicious file detection is that they often suffer performance degradation when class imbalances are present in the training data. For example, Adaboost.M1 is an ensemble of weak classifiers that are combined by majority vote to form a strong classifier. Each weak classifier in the ensemble is trained sequentially such that, on every subsequent iteration, the previously misclassified samples are given more weight for the decision boundary of the current classifier. Obviously, if the data are largely imbalanced, the weighting is often disproportionately given to the majority class. Due to the process of collecting the training data, obtaining normal HTML files is significantly easier than obtaining malicious files and so a significant imbalance exist in our data. For example, our current dataset consists of 7,303 files of which

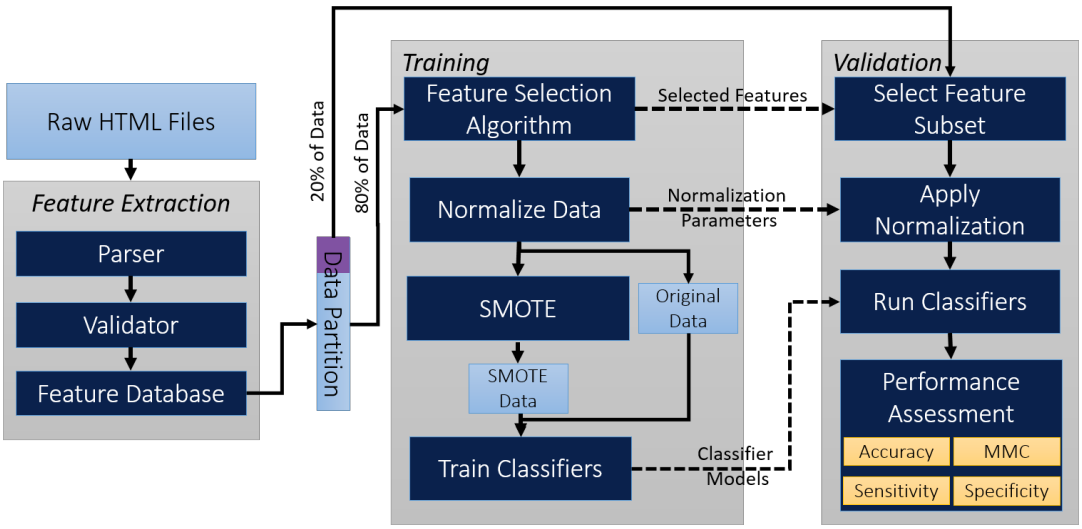

Fig. 8. DAE data flow diagram to analyze HTML files.

7,263 are normal and 40 are labeled as malicious.

One method to resolve the issue of class imbalance for ensemble classifier training is known as Synthetic Minority Oversampling Technique (SMOTE) [6]. SMOTE randomly chooses a point in the minority class (malicious sample) and determines the k nearest neighbors also in the minority class based on the Euclidean distance in the feature space (where k is a user defined parameter). It then randomly chooses one of those k nearest neighbors and it produces a synthetic sample at a random location along the line connecting the originally selected sample with its neighbor. This sample is then added to the minority class and the steps for generating synthetic samples are repeated until the desired number of minority samples have been created. Note that, to insure the features are not disproportionately weighted based on their magnitudes, it is important that the features have gone through a normalization step prior to generating the synthetic samples. Obviously, since the one-class SVM and Isolation Forest use only the majority class (normal files) SMOTE is not necessary for these methods.

In addition to SMOTE, we also use χ^2 feature selection [7]. χ^2 feature selection is a pairwise statistical test to assess how dependent two features are. A user defined probability parameter is set to define the threshold under which the hypothesis of independence between two features can be rejected. If this hypothesis is rejected the feature is not considered unique and is removed from the feature set.

Additionally, other preprocessing steps were experimented with such as dimensionality reduction (e.g. principal component analysis), and/or polynomial feature expansion. However, none of these methods were used in our final framework. Dimensionality reduction was seen as unnecessary as the set of features is relatively low (i.e. 32). Conversely, polynomial feature expansion quickly expanded the dimensionality of our feature set. Neither of these methods showed a noticeable difference in performance and are mentioned here for completeness, they are not presented in our results.

IV. Experimental Results

A. Experimental Protocol

We report our result using 5-fold cross validation. We perform the comparison using the following metrics: accuracy, sensitivity, specificity, and the Matthew's Correlation Coefficient (MCC). If true positives, true negatives, false positives, and false negatives are represented by TP, TN, FP, and FN respectively then the metrics are mathematically represented in the following set of equations:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

$$Sensitivity = \frac{TP}{TP+FN}$$

$$Specificity = \frac{TN}{TN+FP}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

The first metric, accuracy, is an used often for scoring balanced datasets. Unfortunately, as the dataset becomes increasingly imbalanced this metric can be somewhat misleading. For our case – where the data ratio is approximately 180 normal files for every one malicious file - an algorithm that would label any file it was presented with as "normal" would miss all of the malicious files but still get an accuracy score of 99.4%. Clearly, this is not "good" performance. For this reason, we looked at two additional metrics: sensitivity and specificity. Sensitivity is the ratio between the number of correctly labeled normal files and the total number of normal files. Conversely, specificity is the ratio between the number of correctly labeled malicious files and the total number of malicious files. In the application of cybersecurity, the detection on the malicious files (i.e. specificity) is of critical importance whereas improved sensitivity has the effect of

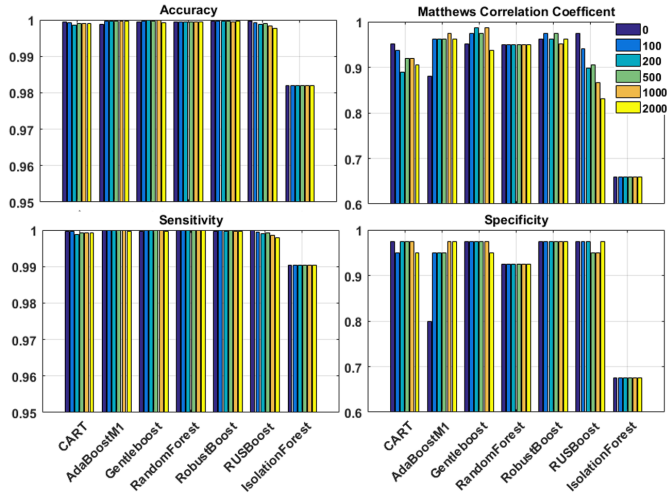

Fig. 9. 5 fold cross validation performance for various classifiers. Accuracy (top left), Matthew's correlation coefficient (top right), Sensitivity (bottom left), and Specificity (bottom right)

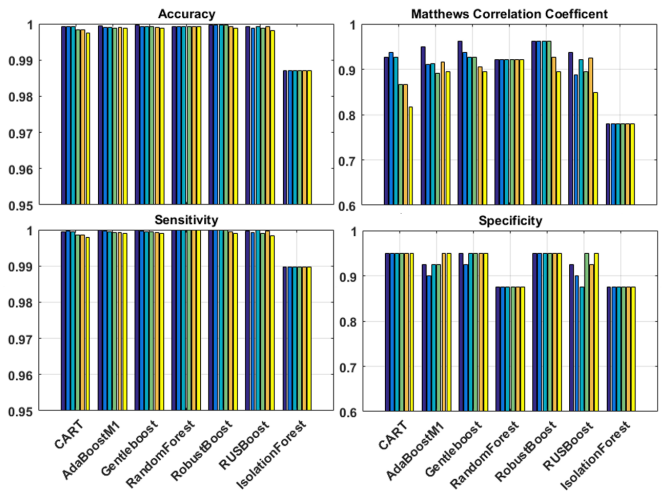

Fig. 10. 5 fold cross validation performance for various classifiers with a reduced set of features obtained from a χ^2 feature selection preprocessing step. Accuracy (top left), Matthew's correlation coefficient (top right), Sensitivity (bottom left), and Specificity (bottom right)

decreasing false alarms. That is to say, from a systems perspective, a slight decrease in sensitivity may be worth a larger increase in specificity. Finally, we used the Matthew's Correlation Coefficient as an overall metric for performance scoring.

B. Results Without Feature Selection

For our dataset of 7,303 total files (7,263 normal and 40 malicious), the 5-fold cross validation process partitions the data into 5 sets of 5842 training samples and 1461 testing samples with equal ratio of normal to malicious files. When applicable, SMOTE was used with a nearest neighbor parameter of 5 and ratio parameter of 0%, 100%, 200%, 500%, 1000%, and 2000%. A parameter of 100% indicates that SMOTE synthetically increases the number of samples by 100%. For example, in our data SMOTE with 100% synthetically increase the 32 minority samples in the training set to 64. The results of the total computed cross-validation accuracy, sensitivity, specificity, and Matthew's correlation coefficient when no feature selection was used can be seen in Figure 9.

We make a couple of observations. First, as expected from the class imbalance, the accuracy performance is high for all cases. To effectively evaluate the relative performance, specificity and the MCC metrics provide more information. From these metrics we observe that SMOTE improves performance in the Adaboost.M1 case and degrades performance in the RUSBoost case. In all other cases the effect of using SMOTE appears to be negligible. It is known that Adaboost.M1 is susceptible to class imbalances so it is expected that SMOTE would improve with a synthetic balancing of the data. However, RUSBoost is a variant of Adaboost.M1 develop to handle class imbalances so it is interesting to see that if the data are synthetically balanced the sensitivity to the majority class can degrade.

C. Results With Feature Selection

Out of the distribution of 40 malicious files, 5 of them were known to be collected from phishing websites. By design, these phishing websites are known to look very similar to normal websites and only contain links to malicious content. Ideally, these 5 samples would be classified as malicious files but the features extracted from the framework presented in this paper do not have sufficient information to distinguish them as such; their features look like normal features. That is, whereas the samples are still technically malicious, they have the effect of being "mislabeled" samples. Thus, the score on the malicious file identification (i.e. specificity) is expected to be no greater than 87.5% (i.e. 35 out of 40). Our results without feature selection (Figure 9) have better scores than our expectation which leads us to conclude that the algorithms are likely over fitting to the minority distribution in the training data.

To reduce the effect of over fitting we ran a χ^2 feature selection algorithm as a preprocessing step. It is expected that, by removing extraneous features from our feature set, over fitting is less likely to occur and the algorithms will improve in robustness. The χ^2 algorithm contains one parameter, p, the confidence threshold on independence between two features. This value was set to 0.05 (meaning at 95% confidence of independence was necessary to keep a feature) which resulted in the six highlighted features shown in Table I being selected.

Figure 10 shows the results as presented in Figure 9 except with the subset of 6 features instead of 32. It is observed that the performance of all the algorithms decreased with respect to the overall MCC metric with the exception of the Isolation Forest. Note that, the dataset still contains the 5 effectively "mislabeled" samples mention previously so, as before, a specificity over 87.5% is unexpected and viewed as over fitting. Interestingly, the Random Forest and Isolation Forest

(the single-class variety of Random Forest) do not appear to over fit with the reduced feature set. Both of these algorithms perform exactly as expected; they missed the 5 malicious files that were indistinguishable from abnormal files.

V. Conclusions

In this paper, we proposed a framework for static classification/detection of malicious HTML files and analyzed the performance of the system using preprocessing and detection algorithms. Furthermore, we also included a sampling method into the framework to address the issue of class imbalance, which is an extremely common problem for HTML classification. Specifically, we analyzed systems the performance on significantly imbalanced data using a decision tree classifier, boosting ensemble classifiers (AdaBoost.M1, GentleBoost, RobustBoost, RUSBoost), Random Forest, and one-class algorithms (i.e., one-class SVM and Isolation Forest). In addition, we have explored the possible benefits of using a preprocessing step to generate synthetic minority samples as well as a feature selection step to reduce over fitting. The benchmark is performed using several skew insensitive statistics as imbalanced data sets are more difficult to evaluate than data sets that are balanced.

Our empirical results show that performance is similar between the different classifiers and the differences are subtle but important. The preprocessing step to generate synthetic minority samples is seen to be beneficial in the case of the AdaBoost.M1 algorithm. In most other cases, the effect of SMOTE appears negligible with the exception of the degradation seen in RUSBoost. More importantly though, our results also show that overfitting is clearly an obstacle for performance when the dataset is imbalanced. Specifically, we have observed that 5 of the 40 malicious samples should have been indistinguishable from the normal files with respect to their features but were often classified as malicious regardless, an indication of over fitting. Even with the introduction of a feature selection process many of the algorithms still over fit the data. The two exceptions were the Random Forest algorithm and its one-class variant, the Isolation Forest.

In the application of malicious detection technology it is often imperative that a detection on a malicious file is not missed. A perfect detection rate can always be achieved but it is at the cost of increased false alarms, which is a significant nuisance to a user. Our empirical findings have shown that many classifier algorithms work well in isolated studies with imbalanced data, providing almost perfect detection with false alarm rates as low as 0.01%. However, our findings also allude to the possibility that some of these algorithms may be over fitting to the data. If the real world malicious data are not well represented by the malicious samples in the training set, it is likely that an over fitted classifier would miss malicious files. In some cases, where robust detection is of paramount importance, it may be best to use an algorithm that learns strictly the distribution of the majority class, such as the isolation forest.

References

[1] M.-S. Pang and H. Tanriverdi, "Security breaches in the us federal government," *Fox School of Business Research Paper*, vol. 17-017, 2017.

[2] A. Bosch, T. Bogers, and M. Kunder, "Estimating search engine index size variability: a 9-year longitudinal study," *Scientometrics*, vol. 107, no. 2, pp. 839–856, 2016.

[3] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[4] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 266–282, 2014.

[5] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[7] H. Schütze, "Introduction to information retrieval," in *Proceedings of the international communication of association for computing machinery conference*, 2008.

[8] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 413–422, IEEE, 2008.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," 1984.

[10] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*, pp. 23–37, Springer, 1995.

[12] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[13] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: Improving classification performance when training data is skewed," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[14] Y. Freund, "A more robust boosting algorithm," *arXiv preprint arXiv:0905.2138*, 2009.

[15] W. Xu, F. Zhang, and S. Zhu, "The power of obfuscation techniques in malicious javascript code: A measurement study," in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pp. 9–16, IEEE, 2012.

[16] S. Yoo, S. Kim, A. Choudhary, O. Roy, and T. Tuithung, "Two-phase malicious web page detection scheme using misuse and anomaly detection," *International Journal of Reliable Information and Assurance*, vol. 2, no. 1, pp. 1–9, 2014.

[17] P. Likarish, E. Jung, and I. Jo, "Obfuscated malicious javascript detection using classification techniques," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pp. 47–54, IEEE, 2009.

[18] Z. Fang, R. Zhu, W. Zhang, and B. Chen, "A half-dynamic classification method on obfuscated malicious javascript detection," *International Journal of Security and Its Applications*, vol. 9, no. 6, pp. 251–262, 2015.

[19] G. Canfora, F. Mercaldo, and C. A. Visaggio, "Malicious javascript detection by features extraction," *e-Informatica Software Engineering Journal*, vol. 8, no. 1, 2014.

[20] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages.," in *SecureComm*, pp. 149–166, Springer, 2012.

[21] P. Satam, D. Kelly, and S. Hariri, "Anomaly behavior analysis of website vulnerability and security,"

[22] G. Qu, S. Hariri, and M. Yousif, "A new dependency and correlation analysis for features," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 9, pp. 1199–1207, 2005.

[23] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural computation*, vol. 12, no. 5, pp. 1207–1245, 2000.