Malicious URL Detection using Machine Learning

D. Cepeda, *20763269*, D. Deane *20788525*, S. Zaki, *20787669*

***Abstract*— In an effort to reduce the negative impact and effects of malicious websites on millions of web surfers' systems, we sought to create a Malicious URL classifier model. This experiment serves as a solution to the problem of identifying malicious web pages thoroughly widely spread throughout the internet. It attempts this by creating, testing and comparing Malicious URL Identification implementations. This is a classification problem in which both typical and custom-made machine learning methods are employed on a data set in order to attempt to accurately determine malicious URLs based purely on a given URL's string contents. To examine this we decided to compare the performance metrics for multiple well known classification methods. These methods include Decision Trees, Random Forests, and Ensemble Voting Techniques.In particular, for this problem, the improvement of the malicious URL class' recall scores, while maintaining relatively high accuracy scores was the main goal set out to achieve. Our experimental results dictate that we were able to achieve our goal by employing numerous approaches explained in detail within this report. The current Malicious URL classifier model is capable of upwards of approximately 90 percent accuracy for each of the individual implementation techniques. The custom-made approach was also able to successfully increase the malicious URLs recall score to approximately 88 percent while maintaining comparable levels of accuracy around 92 percent.**

***Index Terms*— URL, Malicious URL, Random Forest, Decision Tree, Classifier, Ensemble, Feature Extraction.**

I. Introduction

In todays technologically savvy world, people are heavily dependent on the internet for everyday tasks. As a result, new and unexplored web pages are visited by millions daily. Consequently, a large portion of the users are unaware of the malicious content existing within some of the many visited sites that are constantly browsed. This makes unsuspecting users susceptible to scams, information hijacking, as well as malicious infections.

According to Bin Liang et al, see [3], for every 90 websites out there, a surprising 29 of them contain malicious code. Many researchers have attempted to solve this by using a static analysis approach for malicious web page detection. This is done by inspecting malicious JavaScript code within the web pages themselves. The results historically have proven to perform well. Using 77 static JavaScript features, a detection rate in the range of 97-99 percent was achieved. Relatively low False Positive and False Negative rates were also achieved in follow up studies as well, see [4]. The problem with this approach is that the nature of malicious attacks is an ever-changing phenomenon. New tricks, hacks, and techniques are created by attackers everyday. This reduces the ability to combat most recent attacks which can be found in the source code of modern web pages. When approached in this manner, this becomes a theoretically never-ending challenge.

In order to address this issue, R. Patil et al, see [2] created a hybrid methodology combining static and dynamic URL analyses to create features better suited for detecting malicious URLs. F. Vanhoenshoven et all, see [1], addresses the classification of malicious URLs as a binary classification problem, by using different sets of feature combinations based on the feature type.

To overcome the limitations of the above approaches in a simplified manner, a purely dynamic analysis of the URL string contents could prove to be a highly effective and safe strategy for detection modern malicious web pages, without having to visit the page and inspect the page's source code itself. Thus, the task to which we applied machine learning techniques was to the construction of a malicious URL identification classifier tested on an available data set. Also, to prevent the need for advanced feature selection techniques to be manually applied, or for the direct involvement of a domain expert, we also implemented an automated Naive Feature Set Extractor for a given feature set. In conjunction with this, a preferential voting technique was implemented that is modifiable to favour a minority class during the training of typical classifiers. These help in streamlining the entire process to adjust for different data sets, feature sets, and classifiers when training the model.

The data itself was taken from Kaggle's data set library and totalled to 420,464 samples. A collection of 62 features were selected to characterize the data points with an additional 8 features implemented to create a total of 70 URL string-based features.

The classification methods employed include Decision Tree and Random Forest classification methods. These were chosen due to their ability to be trained for classification and prediction with high speed while managing relatively high quality results among leading classifiers. They performed particularly well for the experiments conducted in the aforementioned literature. [1]

The remainder of the report is broken down into the following sections outlining what was done in order to achieve our results. Section II examines, in depth, the approaches that were used for the experiments described in the literature. These were highlighted and used as guidance for our implementation of our project's model. Section III specifies the problems encountered that were necessary to tackle and overcome. Section IV outlines the various approaches we took when creating the functional models. Section V is dedicated to displaying and describing the final experimental results. And, finally, Section VI closes the report by stating conclusive observations and final notes regarding the experiments.

II. Literature Review

A. Paper I: "Detecting Malicious URLs using Machine Learning Techniques" by Frank Vanhoenshoven et al.

This paper addresses the detection of malicious URLs as a binary classification problem. It aims to study the performance of various well known classifiers when employed to detect malicious URLs. The paper focuses on seven classifiers: Naive Bayes, Support Vector Machines, Multi-Layer Perceptron, C4.5 Decision Trees, C5.0 Decision Trees, Random Forest, and k-Nearest Neighbors. The data set used in the literature comprises of 2.4 million URL samples and 3.2 million features, see [1].

Furthermore, the methodology employed in this paper involves splitting the data set into three separate feature sets: the first

consisting of a combination of real-valued and binary-valued features, the second consisting only of binary-valued features, and the third consisting only of real-valued features. The seven classifiers were trained using 2.5 percent of the data set where 121 non-stratified independent random training sets with equal probability were sampled, see [1]. The training was then performed for every type of feature set to compare general performance. This is done based on three performance metrics: classification accuracy, precision, and recall.

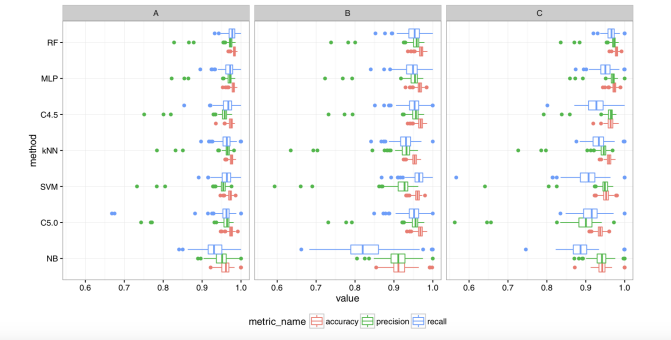

Fig. 1. Accuracy, Precision, and Recall per Classification Method and per Feature Set [1]

Figure 1 shows the results obtained in terms of accuracy, precision, and recall per classification method and per feature set. According to the results, it was concluded that Random Forest is the most appropriate algorithm for this particular classification problem, followed by Multi-Layer Perceptron. As shown in Figure 1, Random Forest achieves the highest scores in terms of both precision and recall, which indicates well-balanced and unbiased predictions by the algorithm. Figure 1 also shows that Feature Set A, which includes both real-valued and binary-valued features outperforms the other feature sets.

All in all, this paper demonstrates that Random Forest and Multi-Layer Perceptron are seemingly the most appropriate classification methods for the detection of malicious URLs, since they attain the highest classification accuracy, precision, and recall.

B. Paper II: "Malicious URLs Detection Using Decision Tree Classifiers and Majority Voting Technique" by Dharmaraj R. Patil, J. B. Patil

This paper provides an effective hybrid methodology for the detection of malicious URLs based on machine learning techniques. Specifically, the methodology involves using different types of Decision Tree classifiers, as well as majority voting of Decision Tree ensembles to obtain high performance for malicious URL detection. The Decision Tree algorithms that were employed include: J48, CART, Random Forest, Random Tree, ADTree, and REPTree. Furthermore, the data set used in this literature was engineered by the authors themselves, in an iterative fashion, in order to obtain higher performance. The data set consisted of 52,082 labelled URL samples and 117 features. It was also balanced in terms of malicious and benign URL counts, see [2].

The framework for the proposed malicious detection system in this paper is shown in Figure 2. The methodology involves a feature extraction phase, a training phase, and a classification phase. Feature extraction is performed for four type-specific feature sets: URL-based, Domain name-based, web page source-based, and short URL-based, see [2]. Following this, the training of six Decision Tree algorithms is performed using the four extracted feature sets in order to select the most appropriate Decision Tree algorithm for each particular feature type. Moreover, an ensemble of various Decision Trees is created, and majority voting is employed in the classification phase to identify whether a URL is malicious or not.

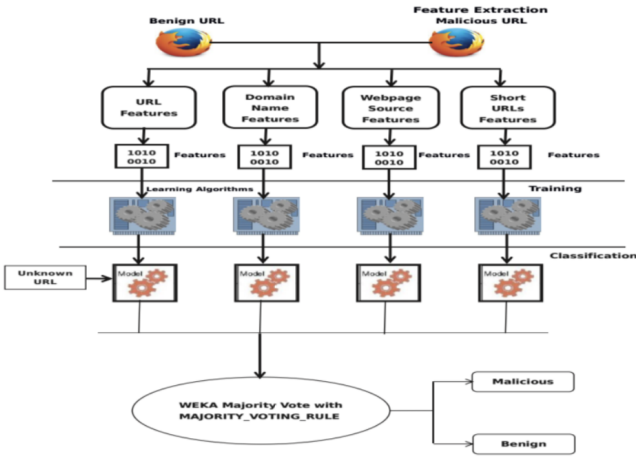

Fig. 2. Proposed Framework for Malicious URL Detection [2]

The evaluation of the classification models implemented as part of this literature is performed considering classification precision, recall, and F-score measures. According to Table 1, which reports the performance of different classification models on different iterations of extracted feature sets, the Random Forest classifier produces the best results with and without the added features. According to the same table, majority voting achieves the second best results with and without the added features.

Classifier	Precision	Recall	F-measure
Without new features			
J48 Decision Tree	0.986	0.985	0.985
SimpleCart	0.984	0.983	0.983
Random Forest	0.990	0.990	0.990
Random Tree	0.979	0.978	0.978
ADTree	0.981	0.980	0.980
REPTree	0.984	0.983	0.983
Majority Voting	0.987	0.987	0.987
With new features			
J48 Decision Tree	0.990	0.990	0.990
SimpleCart	0.992	0.992	0.992
Random Forest	0.994	0.994	0.994
Random Tree	0.982	0.982	0.982
ADTree	0.985	0.985	0.985
REPTree	0.992	0.992	0.992
Majority Voting	0.993	0.993	0.993

TABLE I
PERFORMANCE ANALYSIS OF MACHINE LEARNING CLASSIFIERS [2]

III. PROBLEM DEFINITION

In our experiments, there was problem with imbalance in our data set which we sought to address. The data set used consists of URL strings and their corresponding labels, malicious and benign, having 420,464 samples, where 344,821 of them are benign and 75,643 are malicious. In order to provide a set of features from which our

models could be trained, it was necessary that we proceeded to extract features from the provided data set.

The extraction of features from the URL strings was based on the paper ”Malicious URLs Detection Using Decision Tree Classifiers” [2], where 8 new features were added, resulting in 22 numeric features, 1 continuous feature, and 47 binary features. These features are extracted by a set of python functions iterating throughout the whole data set generating a data frame which can be processed by our models.

The extracted data set consists of lexical features from the URLs strings in addition to binary features representing the presence of selected keywords. The lexical features are textual properties of the URLs themselves, such as length of the string or the count of occurrences of certain characters. The lexical features require prior knowledge of the URL queries which lead to individual counts of predefined special characters. In contrast, the presence of words takes a more subjective approach where the assumption is made that these words may indicate suspicious behavior. Taking this into account, most of the selected words consist of extensions for executables or words related to the access of accounts such as login. The feature set used can be seen in Table 2 below.

No.	Feature Name	Type
1	Length of URL	numeric
2	Length of Query string	numeric
3	Number of Tokens	numeric
4	Number of (!)	numeric
5	Number of (.)	numeric
6	Number of (-)	numeric
7	Number of (_)	numeric
8	Number of (=)	numeric
9	Number of (/)	numeric
10	Number of (?)	numeric
11	Number of (;)	numeric
12	Number of (’(’)	numeric
13	Number of (’)’)	numeric
14	Number of (%)	numeric
15	Number of (&)	numeric
16	Number of (@)	numeric
17	Number of (~)	numeric
18	Number of (+)	numeric
19	Number of (*)	numeric
20	Number of (#)	numeric
21	Number of ($)	numeric
22	Number of Digits	numeric
23	Entropy of URL	continuous
24	Presence of IP address	binary
25	Presence of ”server”	binary
26	Presence of ”client”	binary
27	Presence of ”secure”	binary
28	Presence of ”account”	binary
29	Presence of ”webscr”	binary
30	Presence of ”login”	binary
31	Presence of ”ebayisapi”	binary
32	Presence of ”signin”	binary
33	Presence of ”banking”	binary
34	Presence of ”confirm”	binary
35	Presence of ”blog”	binary
36	Presence of ”signon”	binary
37	Presence of ”login.asp”	binary
38	Presence of ”login.php”	binary
39	Presence of ”login.htm”	binary
40	Presence of ”.exe”	binary
41	Presence of ”.zip”	binary
42	Presence of ”.rar”	binary
43	Presence of ”.jpg”	binary
44	Presence of ”.gif”	binary
45	Presence of ”viewer.php”	binary
46	Presence of ”link=”	binary
47	Presence of ”getImage.asp”	binary
48	Presence of ”plugins”	binary
49	Presence of ”paypal”	binary
50	Presence of ”order”	binary
51	Presence of ”dbsys.php”	binary
52	Presence of ”config.bin”	binary
53	Presence of ”download.php”	binary
54	Presence of ”.js”	binary
55	Presence of ”payment”	binary
56	Presence of ”files”	binary
57	Presence of ”css”	binary
58	Presence of ”shopping”	binary
59	Presence of ”mail.php”	binary
60	Presence of ”.jar”	binary
61	Presence of ”.swf”	binary
62	Presence of ”.cgi”	binary
63	Presence of ”.php”	binary
64	Presence of ”abuse”	binary
65	Presence of ”admin”	binary
66	Presence of ”.bin”	binary
67	Absence of ”www”	binary
68	Presence of ”personal”	binary
69	Presence of ”update”	binary
70	Presence of ”verification”	binary

TABLE II
USED FEATURE SET

IV. METHOD-AND-APPROACH

A. Handling Imbalanced Data

The methods used to handle the imbalanced data include random up-sampling using imbalanced-learn’s RandomOverSampler function. Along with this, an artificial up-sampling technique was also implemented using imbalanced-learns Synthetic Minority Over-sampling Technique function for Nominal and Continuous data.

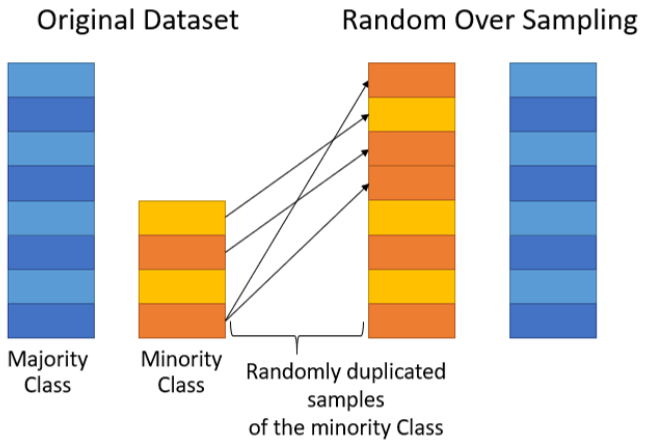

Fig. 3. Random Over Sampling Representation.

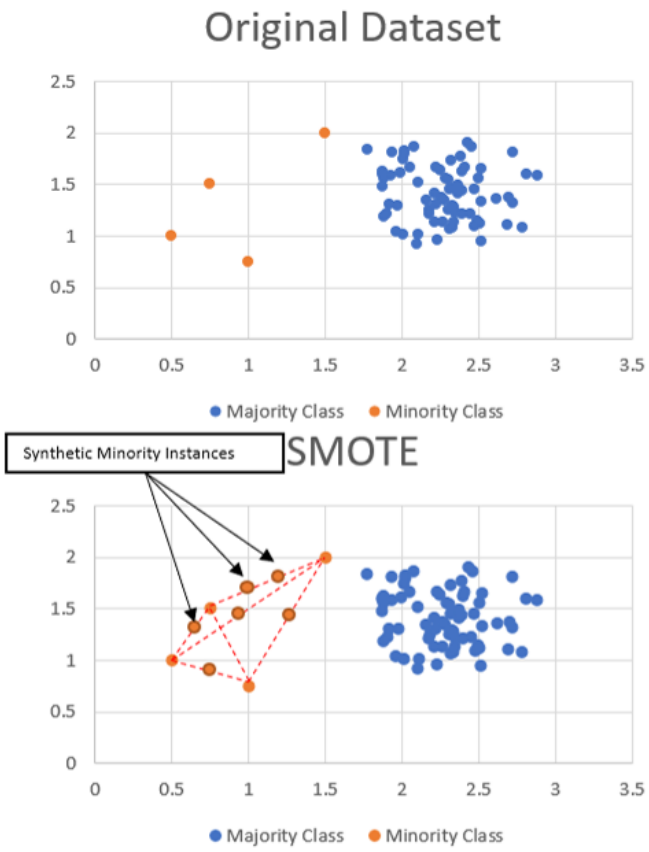

Fig. 4. Synthetic Minority Oversampling Technique Representation.

The goal of using these techniques is to up-sample the minority class of URLs labelled as malicious to equal the frequency of the majority class of URLs labelled as benign. This balancing technique was utilized in place of down-sampling largely due to the consequences associated with applying down-sampling to imbalanced data sets. According to A. Fernndez et. al, down-sampling discards a lot of potentially useful data that could be important for the induction process, see [5]. Along with this, the vast majority of our overall data set would be discarded in the process of down-sampling. This would have lead to the classification models being trained on a significantly reduced portion of URL samples. And thus, this would negatively affect our overall performance as the classifier would have been exposed to significantly less data. This is due to the fact that fewer patterns in the data would likely be found giving way to the overall performance decline.

If, however, our data set was larger and more balanced initially we could have employed a down-sampling technique without having to abandon a substantial set of valuable information. When we employed our up-sampling techniques, it is important to note that we were careful to up-sample only the training set of our data rather than the entire data set itself. This was done judiciously to ensure that the test set was not influential in the feature set extraction, preferential voting, nor classifier training steps. This would have lead to misleading results with respect to the classification methods' performance.

B. Classification Techniques Utilized

1) Decision Tree

The first classification technique we utilized was Decision Trees, a very widely implemented machine learning classifier. Decision Trees take the form of tree like graph structures in which every node in the tree is representative of a test criteria on which each sample in the data set passes through. The criteria used to process these samples are feature attributes. Within the tree structure each branch is representative of the outcome of the criteria test whereas each leaf in the tree represents the resultant label given to a sample based on it's traversal through the tree's criteria. This traversal from the root to a lead is can be described as the classification rules through which samples must pass through in order for the tree to come to a classification decision. This structure seeks to classify the data according to the test criterion correctly while maintaining the comprehensibility and readability of the model's tree structure.

2) Random Forest

Building upon the decision tree is our second and most used classification technique. This machine learning model is known as Random Forests. Random Forests essentially represent an ensemble of decision trees implemented in conjunction with each other in an effort to reduce the negative aspects associated with individually constructed Decision Trees. Random Forests typically are an even stronger modelling technique as the combination of multiple decision trees reduces overall bias and the increased likelihood of overfitting when depending on a single tree structure. This is achieved by creating numerous trees which only include a random subset of the feature set so as to reduce prediction's dependency on any individual feature or set of features.

The classification methods we chose to implement are as follow:

- Random Forest with the unaltered and imbalanced data set
- Decision Tree with a randomly up-sampled training set
- Random Forest with a randomly up-sampled training set
- Random Forest with a synthetically up-sampled training set

In addition to this, we also created a customized method in an effort to outperform these standard up-sampled classification models. This method incorporates the Naive Feature Set Extractor (NFSE) and the Preferential Class Voting Technique which are described in detail later on in this section. The comparative results for each of these classification methods can be seen in Section V.

C. Naive Feature Set Extractor

The Naive Feature Set Extractor (NFSE) method is a simple approach to extract feature combinations that improve a specified classification metric for a given classifier and a chosen class. The algorithm receives a specified classification metric and its minimum required score, a classifier, the size of the subset of features that will be randomly selected, and the amount of randomly generated subsets it is expected to find.

We name the randomly generated subsets of features as filters which determine the features to be kept and the features to be discarded for each feature set. It is important to note that, in order to extract the features from the data set, only the training data should be fed to the NFSE. This is due to the fact that providing the full data set will bias the results of the feature selection and misrepresent the effectiveness of our final classifier, meaning that it was given access to qualities of the test data set that should not be seen by our classifier until the final predictions are made. This follows the same reasoning as to why we excluded the test data during the up-sampling process as explained previously.

NFSE commences by splitting the given data set into a training and a validation set. In this specific implementation, the NFSE incorporates the random up-sampling technique to handle any imbalance found between classes. It should be noted that this could be replaced by other up-sampling techniques such as SMOTE if desired. After this, the model is trained and evaluated if the selected classification criteria meets the defined threshold. The resultant randomly generated filters are then saved and stored. The entire process then repeats these operations until the desired set of filters which meet the specified criteria are found. Figure 5 below shows an overview of the workflow for this implementation.

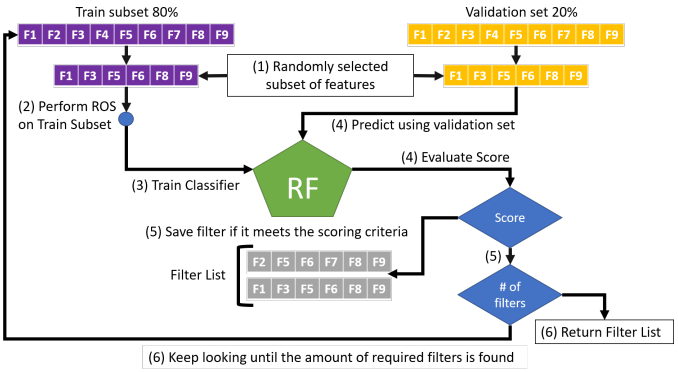

Fig. 5. Naive Feature Set Extraction algorithm.

D. Preferential Class Voting

The preferential class voting technique refers to a soft voting ensemble method in which one class is favoured over another by having at least one more classifiers that is biased to enhance results for a selected class. In our case, this was the malicious class.

The term soft voting simply refers to obtaining the average of the classifiers' predictions in order to generate the final result of the ensemble.

In our implementations, NFSE is used to generate 3 filters which favour the malicious class and 2 filters favouring the benign class. These filtered feature sets are then used to train and predict on an unseen set of data according to the random forest classifiers that are supplied to the soft voting function in order to generate the predictions. See Figures 6 and 7 below for a detailed view of the explained technique.

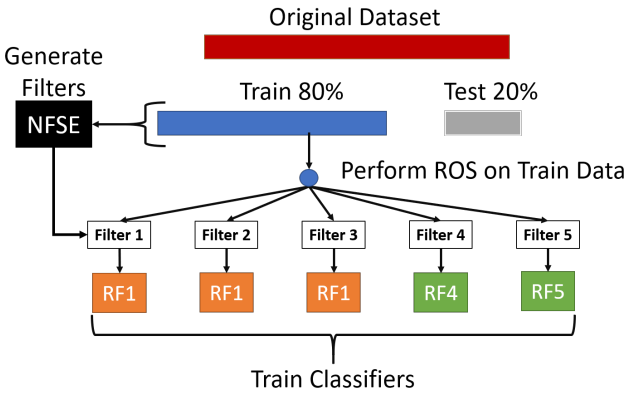

Fig. 6. Training of the Ensemble

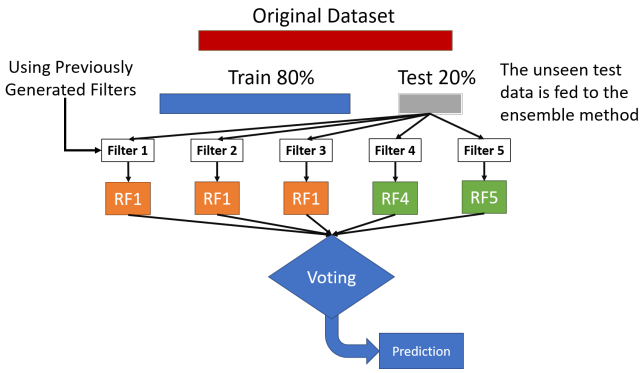

Fig. 7. Testing the Model

V. Results and Discussion

Local comparisons were made between the aforementioned classifiers. For Decision Tree we used the random up-sampling technique. For Random Forest we used a default imbalanced data set and the previously mentioned up-sampling techniques, as well as NFSE with Preferential Voting. By doing this we were able to see the difference in performance between each of the methods.

For the Preferential Voting Technique, the ensemble of five voters consisted of three classifiers favouring Malicious class versus two classifiers favouring for the Benign class. The minimum recall parameter for the "Malicious" URL class was set to 0.815, and the number of randomly kept features was set to 56. On the other hand, the minimum recall parameter for the "Benign" URL class was set to 0.93, and the number of randomly kept features was set to 56.

The results obtained from collective testing are shown in Figure 8 below:

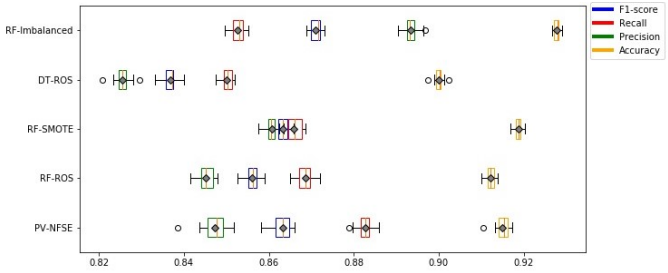

Fig. 8. Classification Report Averages

Generally speaking, all of the classification techniques achieved an average of 90 percent accuracy and upwards. The Naive Feature Extractor in conjunction with our Preferential Voting technique also managed to achieve above 90 percent accuracy while also achieving greater than 88 percent recall score and greater than 84 percent precision on average. This result strikes a good balance between optimizing recall values relative to other approaches while maintaining relatively similar precision scores and accuracy scores.

For the individual class' classification reports, the results depict a similar pattern relative to other classification and imbalance handling techniques.

For the Malicious Classification report the spike in recall as well as the drop in precision for the Naive Feature Selector is more pronounced.

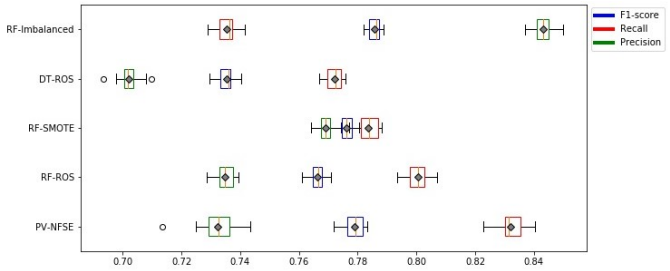

Fig. 9. Malicious URL Classification Report

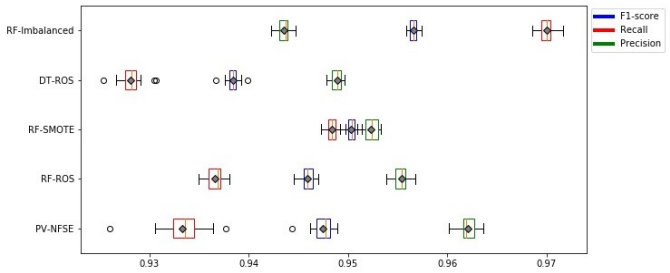

Fig. 10. Benign URL Classification Report

Contrastingly, for the Benign Classification report, we can see that the recall for benign URLs is low and the precision relative to other URLs is high. An expectedly contrasting outcome when compared to the Malicious classification results and the average overall classification results.

We found the suitability of our Naive Feature Set Extractor to be high due to the fact that we sought to generate high recall values while maintaining precision values similar to those of the other techniques. This can be seen in Figure 9 above. It can also be seen that in an effort to increase recall, we pay by the loss of some precision. Generally speaking, while attempting to improve a classifier's recall score, the overall precision is negatively impacted. In our case, the cost of generating false negatives is higher than the cost of generating false positives. This means that the cost associated with classifying a URL which is Malicious as Benign is higher than the cost of classifying a URL which is benign as malicious. This is because our intent was to focus on identifying the genuinely malicious URLs. Thus, we chose to increase recall at precisions at the expense and employ our method. This explains why we set our Naive Feature Extractor and our Preferential Voting Technique to favour recall scores over precision scores which are reflected in the results.

VI. Conclusion

In conclusion, based on our specific goal, we were able to attain the best overall performance in terms of classification accuracy, precision, and recall for detecting malicious URLs when compared to other existing machine learning approaches. All approaches employed were capable of achieving an overall accuracy centered around 90 percent. We were able to develop a customized approach for increasing the recall score of the malicious URL class while maintaining precision and accuracy when compared to standard methods, which was one of the main goals of this project since we were dealing with a highly imbalanced data set.

The data set used was, in fact, one of the biggest limiting factors in this project. The data set had too few samples in order to provide sufficient learning for the different classification models employed in this project when compared to those used in the literature. The data set was also highly imbalanced, with 82 percent of the URL samples being benign URLs, which results in learning bias towards the benign URL class by some algorithms. Another limitation with this project is the long training time required by some of the used algorithms, in particular, our customized approach of NFSE in conjunction with preferential voting.

Some of our proposed future plans geared towards improving the results obtained in this project include enhancing our approach with a more suitable data set. That is, a more balanced data set including more samples, while also extracting additional meaningful features that could be used to produce more patterns to learn from for this particular application. Given processing time limitations, we were unable to apply SMOTE as an alternative up-sampling technique for the minority class before applying NFSE with preferential voting. The results of applying this would determine our next steps as it relates to making advancements with this project. By having such results, we would be able to further analyze the significance of our custom implementations, and in doing so, potentially reinforce that our implementations have consistent behaviour that meets our specific goal.

References

[1] F. Vanhoenshoven, G. Napoles, R. Falcon, K. Vanhoof and M. Koppen, "Detecting Malicious URLs using Machine Learning Techniques," 2016. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7850079&tag=1

[2] D. R. Patil and J. B. Patil, "Malicious URLs Detection Using Decision Tree Classifiers and Majority Voting Technique," 2018. [Online]. Available: https://www.researchgate.net/journal/1311-9702_Cybernetics_and_Information_Technologies

[3] Liang, B., J. Huang, F. Liu, D. Wang, D. Dong, Z. Liang. Malicious Web Pages Detection Based on Abnormal Visibility Recognition. In: Proc. of International Conference on e-Business and Information System Security (EBIS'009, Wuhan, 2009, pp. 1-5.

[4] Patil, D. R., J. B. Patil. Detection of Malicious JavaScript Code in Web Pages. Indian Journal of Science and Technology, Vol. 10, 2017, No 19, pp. 1-12.

[5] A. Fernndez, S. Garca, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera. "Learning from Imbalanced Data Sets." SpringerLink. Accessed March 29, 2019. https://link.springer.com/book/10.1007/978-3-319-98074-4#authorsandaffiliationsbook.