Automated Detection of Phishing Attacks using Machine Learning Techniques

Eric Waters
eswaters@asu.edu
School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona, USA

ABSTRACT

Despite conducting an abundance of sensitive financial transactions and personal business on it, the internet is not always a safe place. Attackers will continue to invent clever ways to gain access to people's credentials and steal their information and/or resources. One popular type of attack is phishing, a type of social engineering where an adversary clones a website and sends the victim a URL to it, anticipating that they will enter their real credentials or other sensitive information into this fake website. These cloned websites often go up and down within days to avoid detection, and there is no single solution to stop it. Furthermore, it is very difficult for authorities to track down the attackers behind phishing websites due to the natural anonymity of the internet. One promising solution is the utilization of data mining techniques to automatically detect and warn users of phishing attacks by looking very closely at the URL that the user is about to click. Since these attackers are mimicking the URLs of legitimate enterprises, they often employ techniques of URL trickery that can be detected by machine learning models that have seen many of these false URLs. Furthermore, the content on these web pages and other attributes of these websites can be inspected by a computer algorithm to gain better insight into their legitimacy. These programs can then warn or even prevent the user from going to a potentially malicious website. This technique is far from perfect, as an attacker may take a novel approach to create their fake URL and slip through the cracks of the phishing detection model. However, if high accuracy and precision can be achieved, it could save thousands to millions of people from becoming victims of phishing attacks.

In this article, I use a dataset containing thousands of phishing and legitimate websites to train several different machine learning models. I then employ techniques to reduce the number of phishing websites that were misclassified as legitimate, as this is the primary scenario to be avoided when creating a system designed to make the internet safer. From the results, I found that the best-performing model was an ensemble of nine different classifiers, where most of the models in the ensemble were tuned to reduce the false positive rate.

1 INTRODUCTION

It is crucial that the internet is safe enough for people to use it for conducting business that involves their personal information such as banking credentials or social security numbers. However, since this information can be used by someone else for their own gain, there will likely always be someone trying to steal it. A popular attack is phishing, a type of social engineering where an adversary clones a website that the victim uses and sends the victim a URL to the clone, anticipating that they will enter their real credentials/information into this fake website. These cloned websites often go up and down within days to avoid detection, and there is no single solution to stopping it. And although phishing is not a new form of attack by any means, its effectiveness is only increasing. In fact, a study by the Ponemon Institute surveyed 591 IT organizations in the US and found that the cost of phishing more than tripled since 2015, increasing from an average cost per business of 3.8 million to 14.8 million in 2021 [3].

There have been many attempts by companies to prevent losses by educating their employees. It is becoming common for companies to conduct "phishing tests", where employees are sent a fake phishing attack attempt. The employees simply are to not click on the URL and are encouraged to report the email. If they do click on the URL, they fail the test and may have to complete phishing training. Given that employees are often made aware that they are subject to phishing testing at any time, one would think that the number of people failing the phishing test would be few and far between. However, Proofpoint's 2021 State of the Phish Report found that the average failure rate on phishing tests was a whopping 11% [9].

This 11% figure may seem absurdly high to someone who is tech-savvy, but the truth is that the average internet user is oblivious to these types of attacks. In 2021 when US workers were surveyed the question "What is Phishing?", only 52% got the answer correct [9]. This is precisely why phishing attacks continue to perform well, despite being one of the older tricks in the book: users of the internet continue to be uneducated about the types of attacks that may be used on them. Companies have tried to educate their workforces, but this can only help so much.

Since educating everybody seems not to be the most effective option, another solution is to develop systems that do the work for the user by either preventing them from clicking on a suspicious link or warning them before they are able to do so. There are currently two primary techniques in place to accomplish this.

The *Blacklist Approach* is where any URL that a user is going to proceed to is automatically compared with a list of known phishing URLs. A fraudulent website would be added to the list as soon enough people report it to be phishing. The clear issue with this approach is that there is a gap in time in which the attacker has effectiveness before the URL is added to the list of known phishing URLs. As new phishing websites are being added frequently, this list cannot possibly cover every phishing website on the internet. Although it is far better than having no protection at all, a better approach is needed.

The *Heuristic Approach* takes characteristics of the website or URL itself in order to guess if it is phishy. This has the advantage over the Blacklist approach in the sense that it is able to determine if a brand-new website is phishy without anyone having to report it. Determining which features to take into consideration and building appropriate models to learn those features determines the effectiveness of this method.

2 RELATED WORKS

The most important aspect of the heuristic approach is the features that are extracted from the URLs and potentially malicious websites. What features are common to most phishing websites? Mohammad et. al [2012] explored these features in detail. They gathered 2500 phishing URLs from the PhishTank [8] archive and used JavaScript and PHP scripts to extract address bar features, abnormal-based features, HTML/JavaScript-based features, and domain-based features. They then analyzed how often each feature appears in the phishing URLs to determine which features were most influential. These 17 features were turned into rules that were then used to create a database of both phishing and legitimate websites for the training of machine learning models to perform automatic phishing detection.

Mohammad et. al [2014b] used the same dataset to train a feed-forward neural network with one hidden layer. This neural network was able to predict with 90-92% accuracy whether or not an arbitrary website is phishing. Mohammad et. al [2014a] also tried a different approach of rule-based classification algorithms such as RIPPER, PRISM, C4.5, and CBA. C4.5 performed the best with an average 94.24% accuracy, while PRISM performed the worst with an average accuracy of 78.76%.

Eventually, Mohammad et. al [2015] created a finalized list of 30 attributes that their experiments had shown were common among phishing websites and could differentiate well between phishing and legitimate websites. Following is a short description of each attribute.

Address Bar Based Features

(1) Using the IP Address as opposed to the domain name.
Example: "http://127.0.0.1/example.html"
(2) Using an unreasonably long URL to hide the details that are suspicious
(3) Using any URL shortening service such as TinyURL
(4) URL containing an "@" symbol. Browsers ignore content before this symbol, so attackers often place their false address after it.
(5) If the characters "//" appear anywhere except directly after the first "http://" or "https://", the user will be redirected to another page.
Example: "http://www.real.com//http://www.fake.com"
(6) Using the "-" symbol.
Example: "http://www.wells-fargo.com"
(7) Having subdomains or multi-level subdomains. Having one subdomain is classified as suspicious, but having two or more is classified as phishy.
Example: "http://www.sites.google.com/view/phishing"
(8) Use of HTTP instead of HTTPs. If the URL does use HTTPS, check to see if the issuer is trustworthy or not. Furthermore, check to see the age of the certificate. Certificates less than two years old are suspicious.
(9) The age of the domain registration. Anything less than one year is phishy.
(10) The favicon (tab icon) being loaded from another domain
(11) Using a non-standard port. Certain ports like those reserved for HTTP (80) and HTTPS (443) should be open, but ports such as 3389 for Remote Desktop features should be closed.
(12) Using "http" anywhere in the domain.
Example: "http://https-www-chase.com"

Abnormal Based Features

(1) The Request URL of objects. The percentage of external objects within the web page that are loaded from another domain is a telling feature because in legitimate websites, most of the Request URLs share the same domain as the web page.
(2) The URLs of anchor HTML tags. In phishing websites, anchor tags mostly redirect to other domains.
(3) Overall percentage of links in meta, script, and link HTML tags. Phishing websites tend to use significantly more links, especially to other domains, within these tags than a legitimate website would.
(4) Server Form Handler (SFH) containing the empty string or "about:blank". This is suspicious because when a user submits a form, the form should do something of significance. Furthermore, if the SFH redirects to another domain, this is classified as suspicious.
(5) Forms triggering the "mail()" or "mailto:" function. Phishers often redirect form submissions to their own emails using these functions.
(6) Abnormal URL. The WHOIS database [11] contains information about the entities registered to a particular domain. Phishing websites often do not have their actual host name included in the URL.

HTML and JavaScript-Based Features

(1) The number of page redirects. Most legitimate websites redirect the user a maximum of one time.
(2) Phishers use a JavaScript trick of changing the status bar on the "onMouseOver" event to show a fake URL.
(3) Disabling right-clicking. Phishers do this to prevent anyone from viewing the source code of the web page.
(4) Using pop-up windows for forms. Legitimate websites will almost never ask you to do this through a pop-up.
(5) Use of IFrames, especially those without borders. IFrames are an HTML tag used to bring one web page into another, and removing the border is a technique used by phishers to clone websites.

Domain-Based Features

(1) The age of the domain. Most legitimate websites have domains older than six months. This information is extracted from the WHOIS database.

(2) The DNS record, which is the system that points a registered domain to the web page's IP address. If the domain has no DNS record, it is considered phishy.
(3) The traffic of the website. If the web page does not rank among the top 100,000 websites by traffic on the database provided by Alexa Internet [1], it is suspicious. If it is not recognized by the database at all, it is phishy.
(4) The PageRank value, assigned by Google Search, attempts to determine how important a website is. The vast majority of phishy websites have no PageRank value, whereas some can reach up to 0.2. Anything below 0.2 is considered phishy.
(5) Whether or not the website appears in Google's index [2] (meaning it can appear in search results). Since phishing websites are often accessible for very short periods, they often are not indexed by Google.
(6) The number of links pointing to the website. Legitimate websites usually have two or more links pointing to them, whereas most phishing websites have zero.
(7) Parties like PhishTank and StopBadware [10] report lists of the most popular phishing domains and IP addresses to avoid. Any website that uses these domains or IP addresses is essentially guaranteed to be a phishing website.

These features have underlying conditions that output a binary or ternary value. For example, the second feature of the address bar-based features, having an unusually long URL, has the following rule:

$$\left\{ \begin{array}{ll} Legimiate\,(1), & \text{if } URL\,length < 54 \\ Suspicious\,(0), & \text{if } 54 \geq URL\,length \leq 75 \\ Phishing\,(-1), & Otherwise \end{array} \right\} \quad (1)$$

These rules are then used to create the dataset which will be used in the tasks outlined in the following section.

3 METHODS

3.1 Data Preparation

Mohammad et. al [2015] published a dataset containing 6,157 legitimate URLs and 4,898 known phishing URLs which have been broken down according to the 30 aforementioned attributes. Being that each attribute is already a binary or ternary value, no cleaning of the attributes themselves was required. However, the dataset contained 5,206 duplicate rows out of the 11,055 total rows. This was initially inflating my performance results because many of the instances in the test set also occurred in the training set. In order to account for this, I partitioned the data such that the testing set contained only unique instances that do not occur in the training set. The training set, 80% of the size of the whole dataset, contains a significant amount of duplicates. These are not duplicate web pages, but rather web pages with the same exact set of binarized features. I decided to leave the duplicates in as I believe they are representative of the real world and not due to an error. It is reasonable that many websites behave the same way or have URLs with the same properties. Furthermore, I thought that perhaps these duplicate rows would help to reinforce the learning of some of the models to be tested. To test this theory, I ran all of my models with the duplicates in and with the duplicates removed, ensuring that in both cases the test set contained unseen instances. The models run with duplicate training data performed significantly better in virtually all regards. Therefore, it does not appear that these duplicate rows led to any overfitting.

3.2 Model Training

Using the Scikit-learn, Pandas, and NumPy Python libraries, I trained the following machine-learning models on the data.

(1) Gaussian Naïve Bayes. I used the default parameters.
(2) Decision Tree. I used the GridSearchCV hyperparameter tuning technique to find that the *entropy* criterion and *random* splitter were the ideal parameters.
(3) Decision Tree with AdaBoost. I again used the *entropy* criterion and *random* splitter on the decision tree. I used *100* estimators.
(4) Random Forest. GridSearchCV revealed that the *entropy* criterion, *log2* max_features, and *150* estimators worked best.
(5) Support Vector Machine. I used a C of *100* to strongly avoid misclassifying training instances.
(6) K-Nearest Neighbors. My hyperparameter tuning indicated that the *l1* distance metric, *distance* weight function, and *10* neighbors were ideal.
(7) Logistic Regression. Tuning showed that the *l2* penalty norm and a C value of *0.1* produced the best results.
(8) Multi-layer Perceptron Artifical Neural Network with 1 hidden layer. My general testing found that a hidden layer size of *256* worked well, along with the *ReLU* activation function and the *Adam* optimizer.
(9) Multi-layer Perceptron Artifical Neural Network with 5 hidden layers. Similarly, I used five layers of size *256*, again with the *ReLU* activation function and *Adam* optimizer.
(10) Voting Ensemble of all of the above classifiers. Using *soft* voting, where the class label is determined using an aggregation of the predicted probabilities from each classifier, outperformed *hard* voting (basic majority vote).

Furthermore, I applied dimensionality reduction techniques to transform the data from 30 dimensions into 10. I then used a Support Vector Machine to compare the performance of these techniques.

(1) Principal Component Analysis. PCA revealed that the first 10 components explain approximately 70% of the variance in the data.
(2) Kernel Principal Component Analysis. I surmised that KPCA may outperform PCA due to the data likely not being linearly separable.
(3) Autoencoder Neural Network. The encoder had *3* dense layers *(128, 64, 10)*, and the decoder also had 3 dense layers *(64, 128, 30)*. I used the *Adam* optimizer and *Mean Squared Error* loss function to train the model for *100* epochs with a batch size of *32*.

My thought process was that if any of these dimensionality reduction techniques could beat the performance of the SVM that did not employ dimensionality reduction, then perhaps dimensionality reduction would be a useful technique to try on other classifiers.

3.3 Minimizing False Positives

Although the dataset is reasonably balanced and therefore accuracy is an important metric, the false positive metric is arguably more important in the case of phishing detection. Please note that the dataset uses "Legitimate" as the positive label and "Phishing" as the negative label. For this reason, it may help to think of this task as "Legitimate Website Detection" as opposed to "Phishing Detection" when discussing metrics such as false positives. That being said, a false positive would indicate that the URL is truly a phishing website, but the model classifies it as legitimate. This is troublesome because if this system is to prevent users from going to phishing websites, the model would tell the user that the phishing website is legitimate and they could proceed to have their credentials or other information stolen. False negatives, on the other hand, are less of an issue. In this case, the website is truly legitimate, but the model would warn the user that this website could be a phishing attack. Although it is still desirable to avoid these false negatives, it is more important to avoid false positives. Naturally, the false positive rates and false negative rates are a tradeoff. Attempting to lower the rate of one will likely increase the rate of the other and vice versa. This is because the model is essentially being told to lean toward one side of caution. With my models, I instructed them to lean towards the side of predicting that the URL is phishing rather than legitimate. I tried this in three ways.

(1) Changing class weights on the machine learning models that support it. These were Logistic Regression, Decision Tree (with and without AdaBoost), Random Forest, and SVM. Changing the class weights means that I am penalizing the model more for misclassifying instances of phishing. The amount I weigh each class varied from model to model. For instance, logistic regression produced the best results when I penalized it 2x as much for misclassifying phishing websites. However, the Decision Tree, Random Forest, and SVM all performed best when I penalized it 100x as much.
(2) Recreating the Voting Classifier using the ML models with class weights applied. For each model that could have class weights applied, I replaced it in the Voting Classifier with the modified version.
(3) Using probabilities to manually change the outputs of certain models. These machine learning models produce a probability of the instance being each class. For example, it might think a given testing point has a 60% chance of being legitimate and a 40% chance of being phishing. Normally, it predicts the class label of the class with the highest probability. However, for the best-performing models, I manually overrode the predicted class label if the probability of the instance being phishing was over a specified threshold. For most models, this threshold was around 0.2, i.e., if there was at least a 20% chance of the instance being phishing, then I classified it as phishing. I used this technique on the Multi-layer Perceptron, Random Forest (with and without class weights applied), and the Voting Ensemble (with and without the class weights applied).

4 RESULTS

For each machine learning model, I evaluated the performance using several metrics.

(1) Overall accuracy. This is the number of correctly predicted test instances over the total number of test instances.
(2) Precision. This is the proportion of instances classified as legitimate websites that were truly legitimate websites. A model with no false positives has a precision of 1.00. Since my primary objective was to avoid false positives (identifying a phishing website as legitimate), I primarily aimed to maximize this metric above all else.
(3) Recall. This is the proportion of legitimate websites that were correctly identified. Since I am training my models to prioritize avoiding false positives, the recall metric will reveal the downside of doing so: the number of false negatives I accumulate as I misclassify legitimate websites as phishing in an attempt to be over-cautious.
(4) F1-score. This is the harmonic mean of the precision and recall and is typically a good overall indicator of a model's performance. However, in the case of phishing detection, I am not too concerned with maximizing the f1-score. This is because I would prefer a model with 0.99 precision and 0.90 recall to a model with 0.96 precision and 0.96 recall. Although the latter may have a higher f1-score and in other applications be considered a better model, in this case, I want to avoid telling users that the website they are visiting is safe when it is a phishing attack in reality. Again, the consequences of false positives are much more drastic than those of false negatives.

Therefore, my primary goal while training these models was to maximize the precision while keeping the recall to a reasonable level. I glanced at the accuracy and f1-score to very quickly assess the model's general performance, but I was not attempting to maximize those metrics. Below in Table 1 are the first 10 models that I trained without any attempts to maximize the precision.

Table 1: Initial ML Classifier Performance Results

Classifier	Accuracy	Precision	Recall	F1
Naïve Bayes	0.63	1.00	0.31	0.47
Decision Tree	0.96	0.97	0.96	0.97
DT + AdaBoost	0.97	0.97	0.98	0.98
Random Forest	0.97	0.97	0.98	0.97
SVM	0.97	0.96	0.97	0.97
K-Nearest Neighbors	0.96	0.96	0.97	0.96
Logistic Regression	0.92	0.92	0.95	0.93
MLP (1 layer)	0.97	0.96	0.99	0.97
MLP (5 layers)	0.97	0.97	0.98	0.98
Voting Ensemble	0.97	0.98	0.98	0.98

Next, I attempted to use dimensionality reduction techniques to beat the performance results of SVM. This was to test if dimensionality reduction would be a worthy path to pursue.

Table 2: DR + SVM Performance Results

Classifier	Accuracy	Precision	Recall	F1
PCA + SVM	0.93	0.93	0.94	0.94
Kernel PCA + SVM	0.93	0.93	0.94	0.94
Autoencoder + SVM	0.91	0.91	0.91	0.91

Table 3 contains the results of the addition of class weights to the models which supported it. An additional column, "penalty", has been added to indicate the penalty applied to the model for misclassifying a phishing website. The last row outlines the results for the ensemble that was a result of replacing the original models with their newly weighted variants. The ensemble itself does not have a penalty applied, so the field is left blank. The optimal penalties were determined by trial-and-error testing.

Table 3: Effect of Class Weights on Performance

Classifier	Penalty	Accuracy	Precision	Recall	F1
LR	2x	0.92	0.94	0.91	0.93
DT	100x	0.96	0.97	0.96	0.97
DT + Ada	10x	0.97	0.98	0.96	0.97
RF	100x	0.97	0.98	0.96	0.97
SVM	100x	0.96	0.98	0.95	0.96
Ensemble	–	0.98	0.98	0.97	0.98

Furthermore, I applied the manual changes outlined in the Methods section. The threshold for switching the class label from "legitimate" to "phishing" is specified in its own column. The optimal threshold was determined by trial-and-error testing.

Table 4: Manually Changing Labels using Probabilities

Classifier	Threshold	Accuracy	Precision	Recall	F1
MLP[1]	0.06	0.96	0.99	0.93	0.96
RF	0.23	0.97	0.99	0.95	0.97
RF	0.10	0.94	1.00	0.89	0.94
RF[2]	0.31	0.97	0.99	0.95	0.97
RF[2]	0.10	0.94	1.00	0.89	0.94
Ensemble	0.29	0.96	0.99	0.95	0.97
Ensemble	0.18	0.94	1.00	0.89	0.94
Ensemble[3]	0.46	0.98	0.99	0.97	0.98
Ensemble[3]	0.24	0.96	1.00	0.93	0.96

[1] 1-layer
[2] Uses class weights (100x penalty)
[3] Uses the modified models with weights (same Ensemble from Table 3)

5 DISCUSSION

The initial models outlined in Table 1 performed above my original expectations. The f1-scores were in the 0.93-0.98 range, with the exception of Gaussian Naïve Bayes. Curiously, Gaussian Naïve Bayes achieved a precision of 1.00, but with a recall of only 0.31. This means that it was being far too cautious and predicting that the vast majority of test instances were phishing if it had any shred of doubt. Despite having a lackluster accuracy of 0.63, I chose to include it in the ensemble models because, in the case of a close call, I would hope that the Naïve Bayes classifier's lean toward the side of caution would push the ensemble to predict that the instance is phishing rather than legitimate. Although the precisions of other models were still relatively good (0.92-0.97), I pushed to improve them further.

Since the data had 30 dimensions, I thought that dimensionality reduction might be worth exploring. The original SVM without any dimensionality reduction had an f1-score of 0.97. Table 2 demonstrates that dimensionality reduction does not result in better performance for this particular dataset, at least with the SVM classifier. The performance was so much worse that I chose not to attempt dimensionality reduction techniques on any other classifier.

In Table 3, I was glad to see improvements in precision across most models. They responded well to the increased penalty for misclassifying instances of phishing. Most models saw a slight increase in their precision, with a slightly higher drop in their recall, which was acceptable.

What defines the best-performing model? Throughout the development of these models, I was attempting to maximize the precision while holding the recall to a respectable value. I hoped to achieve a precision of 0.99 or 1.00, which is why in Table 4 the thresholds are inconsistent. For each of the Random Forest and Ensemble classifiers in Table 4, I used trial-and-error to find the lowest threshold that would reach a precision of 0.99 and subsequently the lowest threshold that would reach a precision of 1.00. Note that there was no threshold that would bring the MLP classifier to 1.00, as it was giving probabilities of 1.00 on too many misclassifications.

I believe that the contest for the strongest model is a toss-up between the last two models shown in Table 4. As a reminder, these models are Voting Ensembles (Soft Voting) consisting of:

(1) Gaussian Naïve Bayes
(2) Decision Tree, penalized 100x more for misclassifying instances of phishing
(3) Decision Tree + AdaBoost, penalized 10x more for misclassifying instances of phishing
(4) Random Forest, penalized 100x more for misclassifying instances of phishing
(5) Support Vector Machine, penalized 100x more for misclassifying instances of phishing
(6) K-Nearest Neighbors
(7) Logistic Regression, penalized 2x more for misclassifying instances of phishing
(8) Multi-layer Perceptron with 1 hidden layer
(9) Multi-layer Perceptron with 5 hidden layers

The first model decided that the instance was phishing if the model believed there was at least a 46% chance of it being phishing. It achieved an accuracy of 0.98, precision of 0.99, recall of 0.97, and f1-score of 0.98. The latter model decided that the instance was phishing if the model believed there was at least a 24% chance of it being phishing. It achieved an accuracy of 0.96, precision of 1.00, recall of 0.93, and f1-score of 0.96. If I had to choose between the two, I would prefer the second, higher-precision model. This

is because it is simply more cautious and sacrifices accuracy to prevent false positives. Out of the 2,211 instances in the test set, it only misclassified 5 instances of phishing and 87 legitimate websites. In practice, this would translate to allowing approximately 0.22% of phishing websites to go undetected and incorrectly warning the user approximately 3.9% of the time that a legitimate website is phishing. I would prefer that to the previous model, which allows about 0.77% of phishing websites to go undetected and incorrectly warns the user about 1.5% of the time that a legitimate website is phishing.

6 CONCLUSIONS

Phishing attacks, despite their age, have only grown in popularity in recent years [3]. As the internet also continues to grow in popularity, more and more users become vulnerable to having their valuable personal information or resources stolen. As a result, there is an increasing demand for highly accurate phishing detection systems that can protect internet users from proceeding to malicious websites. Although not the only solution, the employment of machine learning techniques is a promising avenue for creating such a detection system.

Related works [6, 7] have used the phishing dataset developed by Mohammad et. al [2012] to train single-layer neural networks and rule-based classifiers. However, some datasets naturally lend themselves to certain machine learning models better than others. It is often worth exploring several options to see which will perform the best. Furthermore, in the case of phishing, special emphasis ought to be placed on preventing false positives, i.e., telling the user that a phishing website they are about to visit is legitimate and allowing them to proceed.

By creating an ensemble of 9 different machine learning models, some of which I tuned to prevent false positives, and then tuning the outputs of the ensemble itself, I was able to achieve a false positive rate of 0.22% while only allowing a false negative rate of 3.9%. This model surpassed my initial expectations and outperformed the other 26 models that I created. I believe that it performed best because it incorporated a bit of each of the techniques that I used to improve overall performance and minimize the false positive rate.

Improving the accuracy and precision of these phishing detection models is vital for creating a safer internet. Furthermore, it can save companies millions of dollars on average annually [3]. These models can then be used in applications where phishing attacks occur, such as browsers or e-mail applications. It is highly recommended that everyone, even technologically sound users of the internet, utilize anti-phishing technology to protect themselves. Akin to wearing a seat belt, it is better to be always protected, even when the user is highly confident in their abilities.

Future work on this topic should and will continue. Although the 30 features developed by Mohammad et. al [2015] have proven effective, they are certainly not every feature that could be considered. Analysis of more aspects of phishing websites that can differentiate them from legitimate ones will serve to increase the performance of phishing detection systems. Furthermore, I anticipate that academic entities and enterprises will both continue to refine their machine-learning models in order to achieve the highest performance possible. And, as new machine learning models or other classification techniques are undoubtedly devised in the future, their newfound strengths should be immediately applied to phishing detection systems in order to provide the best protection for internet users.

REFERENCES

[1] Alexa Internet 2022. https://www.alexa.com/ Accessed: 2022-05-01.
[2] Google Index 2022. https://support.google.com/programmable-search/answer/4513925?hl=en#:~:text=The%20Google%20index%20is%20similar,and%20updates%20the%20Google%20index. Accessed: 2022-11-30.
[3] Ponemon Institute LLC. 2021. *The 2021 Cost of Phishing Study.* Proofpoint, Inc.
[4] Rami Mohammad, Fadi Thabtah, and TL McCluskey. 2015. Phishing websites dataset. (2015).
[5] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. 2012. An Assessment of Features Related to Phishing Websites using an Automated Technique. *International Conference For Internet Technology And Secured Transactions* (2012), 492–497.
[6] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. 2014. Intelligent rule-based phishing websites classification. *IET Information Security* 8, 3 (2014), 153–160. https://doi.org/10.1049/iet-ifs.2013.0202 arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-ifs.2013.0202
[7] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. 2014. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications* 25, 2 (2014), 443–458.
[8] PhishTank 2022. http://www.phishtank.com Accessed: 2022-11-25.
[9] Inc Proofpoint. 2021. *State of the Phish: At a Glance.* Proofpoint, Inc.
[10] StopBadware 2022. https://www.stopbadware.org/ Accessed: 2022-11-30.
[11] WHOIS 2022. https://who.is/ Accessed: 2022-11-29.