Lucas Huang, Sean Lee, Alex Huang, Helen Lyons

CPSC 381: Introduction to Machine Learning

May 8th, 2024

Phishing Attack Detection with Machine Learning Methods

Alex Huang, Lucas Huang, Sean Lee, Helen Lyons

Introduction

The proliferation of phishing attacks poses a significant threat to individuals, businesses, and organizations worldwide. According to the FBI's 2022 Internet Crime Report, phishing was the most common type of cybercrime reported, affecting hundreds of thousands of victims and amounting to a total dollar loss of $52 million annually. Besides these hefty monetary losses, phishing scams lead to massive breaches of private information and malware installation. In recent years, phishing techniques have become more sophisticated, making it more difficult for traditional security measures to detect phishing attempts; for example, spear phishing is a technique which involves highly personalized and convincing emails. Since phishing exploits human psychology rather than technological vulnerabilities, it is important to develop detection tools to warn people about potential security risks in order to improve worldwide internet safety.

The main goal of this project was to classify websites into phishing and non-phishing categories by utilizing machine learning methods. Our approach was twofold: first, our main analysis consisted of classifying websites based solely on their URLs, since this is likely how people would evaluate websites when confronted with potential phishing emails. From a dataset consisting of URLs and their corresponding label indicating whether they are safe or unsafe, we extracted features from each URL such as its length, number of special characters, and the presence of code to use as input for a variety of classification machine learning models.

Our secondary analysis was a more in-depth project involving a full investigation of the request to the website. This project involved an examination of each available website's HTML

code and associated content, such as headers, and cookies to detect more sophisticated phishing attempts which may not be apparent from solely the URL.

Finally, we attempted some basic adversarial attacks for fun - we tried to craft phishing inputs that the model classifies as legitimate. This allowed us to better understand the limitations of our model and how it works. After examining some graphs and figuring out what influences the model the most, we found that increasing the length of any arbitrary phishing URL (with alphabetic characters) significantly increases the chance that our model marks it as safe (which, in practice, would be very easy to do for any phishing URL).

Some primary challenges that we faced were deciding which features to extract from URLs and which to keep in our final model. There were many tradeoffs we had to make in terms of speed of extracting certain features and how much complexity they would add to the model (some which increased our training time to an unreasonable degree). We ultimately decided on a relatively small set of mostly numerical features, as well as some categorical flags for certain features we thought would be useful in distinguishing URLs (e.g. presence of code/script tags, URL shorteners, etc.)

Data

We obtained our data from Kaggle's "Malicious URLs" dataset, which consists of 650,000 URLs that are either benign links, defacement (hacked) links, and phishing links. Of these 650,000 initial URLs, 20% were classified as phishing, and 80% non-phishing. From here, we filtered out 40% of links by removing all inactive websites that did not respond to HTTP requests; after this removal, only 3% of URLs were phishing URLs (which makes sense, since phishing links get removed frequently, DNS providers are usually temporary sites, etc.) From

here, we extracted textual features from the URLs themselves as well as by pinging all the links to gather more data about the website itself and data from the HTTP request.

Methodology

We classify URLs using three major machine learning methods: Logistic Regression, Stochastic Gradient Descent with Hinge and Perceptron loss, and Random Forests. Random Forests is an ensemble learning classifier that generates a set of decision trees using random subsets of data and features, and then classifies data points by averaging predictions over those trees. The algorithm uses bootstrap sampling with replacement to select a subset of training data for each tree, and subsequently selects a subset of random features to split the nodes of each tree. The use of multiple decision trees restricted to different training sets and features reduces variance and overfitting.

For each of these methods, we trained models using a core set of 19 features as well as an expanded set of 24 features.

Implementation Details

We begin data preprocessing by separating benign and defacement links from phishing links. Then, for our primary analysis, we extract 19 numeric features:

- Length of URL
- Number of alphabetic characters
- Number of numeric characters
- Number of specific special characters: “@”, “#”, etc. (12 types of characters)

As well as several categorical features:

- Presence or absence of an embedded IP address
- Whether the URL is in shortened form (e.g. goo.gl)

- Presence of “HTTPS” in URL
- Presence of code script tags in URL

We implement our Logistic Regression, Stochastic Gradient Descent, and Random Forests Models using the sklearn package. We trained our SGD model with both a Perceptron and Hinge loss. The Random Forests package defaults to using 100 trees; since we have hundreds of thousands of data points, we also trained a model using 500 trees. For both of these models we used the default parameter of floor(sqrt(19)) = 4 features per tree, and declined to enable a maximum tree depth or minimum number of samples per leaf.

In our secondary analysis, we collect more data by sending out http requests to every link. We coded a web scraper with the aiohttp package to comb through the urls in parallel and appropriately label samples that had problems connecting. We filtered out any samples that had errors connecting including SSL errors, DNS errors, Timeout errors, etc. We run the same analysis as before with five additional features:

- Length of response content
- Whether the returned request is an html file
- Number of headers
- Number of cookies
- Response Status Code

Results

We sailed high over the bar we set for ourselves at the onset of the project. While we initially declared that any classifier that predicted phishing links better than random chance would be considered a success, we have developed a classifier with true potential for future investigation. Out of the three models, Random Forests was the strongest, since it had the highest overall accuracy rate as well as the lowest false positive and false negative rate — our results are printed below:

Logistic Regression
Logistic Regression Accuracy: 0.8394720475433626
Percentage of phishing URLs not caught: 0.7878740095399535
Percentage of benign URLs accidently caught: 0.008782134411473033

Stochastic Gradient Descent
SGD Classifier Accuracy (Hinge Loss): 0.8418215741828484
Percentage of phishing URLs not caught: 0.7936457881666601
Percentage of benign URLs accidently caught: 0.0044531324496996285

SGD Classifier Accuracy (Perceptron): 0.7587972880627155
Percentage of phishing URLs not caught: 0.4047853679687808
Percentage of benign URLs accidently caught: 0.20163059025460092

Random Forests
RandomForestClassifier Accuracy (100 Trees): 0.920054668724422
Percentage of phishing URLs not caught: 0.2843458036031064
Percentage of benign URLs accidently caught: 0.03054199404988939

RandomForestClassifier Accuracy (500 Trees): 0.9200239559578928
Percentage of phishing URLs not caught: 0.2836362202861986
Percentage of benign URLs accidently caught: 0.030675490121290717

As is evident from the metrics, we determine that both Logistic Regression and SGD yield results with decent accuracy but very high false negative rate, which in our case is dangerous for potential users — so we discard these methods. On the other hand, our Random Forests model has a significantly lower false negative rate (28.4%), as well as higher overall accuracy. Random Forests does have a higher false positive rate than the other two models (3.0%), but this is less relevant since false positives are much less dangerous to users than false negatives, and also 3% is still relatively good.

We also tried attacking the model by trying to get it to classify phishing URLs as safe. We found that increasing the length of the URL itself increases the chance that a URL is classified as safe. This is explained by the double boxplot of safety compared to URL length we made–we can see how safe URLs tend to be longer in general than unsafe ones, and a majority of the outlier long URLs are safe ones. Trying this trick on a random sample of unsafe URLs in our test dataset, 63% of them are able to be converted to safe URLs with this trick, which is one of the limitations of our model.

Below are our results from our secondary analysis:

HTTP Request Data Analysis

Logistic Regression
Logistic Regression Accuracy: 0.964561500275786
Percentage of phishing URLs not caught: 0.8409893992932862
Percentage of benign URLs accidently caught: 0.0027263595924809873

Stochastic Gradient Descent
SGD Classifier Accuracy (Hinge Loss): 0.9450496414782129
Percentage of phishing URLs not caught: 0.9796819787985865
Percentage of benign URLs accidently caught: 0.017398478978332615

SGD Classifier Accuracy (Perceptron): 0.9418436293436293
Percentage of phishing URLs not caught: 0.9726148409893993
Percentage of benign URLs accidently caught: 0.02102166738412972

Random Forests

RandomForestClassifier Accuracy (100 Trees): 0.9928985107556536
Percentage of phishing URLs not caught: 0.17314487632508835
Percentage of benign URLs accidently caught: 0.0006098435930549577

RandomForestClassifier Accuracy (500 Trees): 0.9926571980143408
Percentage of phishing URLs not caught: 0.16784452296819788
Percentage of benign URLs accidently caught: 0.0005739704405223131

The inclusion of these five features brings our Random Forests classifier up to 99% accuracy — which is far above the 50% accuracy we would expect from a random chance classifier. In addition, it significantly decreases our false positive and false negative rates to 0.05% and 16.7% respectively, making this a safer model for potential future use. Looking at the distribution of content length, cookies, and headers, we can see some general trends. In general, safe links had longer content, more cookies, and significantly more headers. This may be due to the increased amount of data real websites need to maintain a website while malicious links may be more minimal in website and server design. Below, we created several boxplots to visualize this: