Phishing Websites Detection Using Python And R

Mehta Karan Hemendra, Tanmay Gupta and Niket Shah

Abstract

Phishing is a challenging, well known security attack. Phishing attacks are used to attain sensitive data of a customer like username, passwords, personal identification numbers PINs number etc. In a phishing attack the target user is tricked into entering his/her sensitive credentials in some fake website created by the attacker. For example, let's assume that an attacker is interested in attaining the target user's Facebook username and password. Then the attacker creates a fake Facebook login page that is an exact replica of the original site. The target user is tricked into visiting the fake Facebook page and entering his/her credential information. Once the target user has entered his/her credentials and hits the submit button the attacker has access to the target users information. The target user can be directed to the fake website in many ways. One of the ways includes sending the target user an email, the contents of which are written in such a way that it look like a legitimate email, so that the spam filter does not classify the email as spam. The content of the email is written with an intention to hook the target's interest. Once the user's interest is piqued he/she is bound to open the link of the fake website, therefore falling into attackers trap.

Keywords: Phishing, Web Security, Facebook Phishing.

---

1. Introduction

Phishing attacks are mostly used to target internet banking applications, online shopping websites (Amazon, Flipkart), gambling website (PartyPoker) etc. Once the attacker has the necessary data, they may misuse the data to empty the user's bank account, blackmail them etc. Preventing phishing attacks presents many challenges. Tackling phishing is tricky since; most of the users are not good at using computers and are illiterate towards the internet. The attacker uses the user's lack of knowledge to his advantage and tricks the victim into unknowingly assisting the attacker. Let's assume that the attacker wants to attain the credentials of the target user for an XYZ website. Broadly speaking these are the steps that the hacker will follow:

- The hacker will study XYZ website and create an User Interface which looks exactly same or almost similar to XYZ.
- Then the hacker will host XYZ website and send the link of the fake website to the target user.
- The target user will receive the URL to the fake site via some email or other means.
- When the target user uses the fake URL to go the fake XYZ website, he/she believes that they have opened the original XYZ website since the UI of the fake website is almost similar to the original website.
- The target user enters his/her credentials and upon submitting the attack is successful.

Our goal is to sort out such websites from a pool of websites and flag them as phished or could be phished so that they can be reported for malpractice under Cyber Security.

2. Research Scope

The main aim of the project is to identify and eradicate websites that are considered to be phished and the websites that could be phished from a data set of websites in irregular formats. To process these websites, they first need to be brought to a common format and then and only then can they be operated upon for phishing detection. The algorithm works by generating a Ruleset which regulates the websites and then flags them as phished or not based upon their attribute values. The algorithm has been implemented in Python as well as R. In both the cases, the accuracy percentage is threshold at 90%, and thus we can confirm that the algorithm implemented is acceptable or not. There are different packages used in both of the languages that help in achieving the implementation of the project. Also, the dataset needs to be large enough to avoid the over fitting or under fitting of the data and subvert incorrect results from the classifier. The dataset used is the UCI Machine Learning Websites dataset, which contains over 2500 tuples and more than 30 attributes for classification.

3. Dataset

3.1. Our Approach

The dataset used in the project is the UCI Machine Learning dataset which is publicly available on the website:

https://archive.ics.uci.edu/ml/datasets/Phishing+Websites

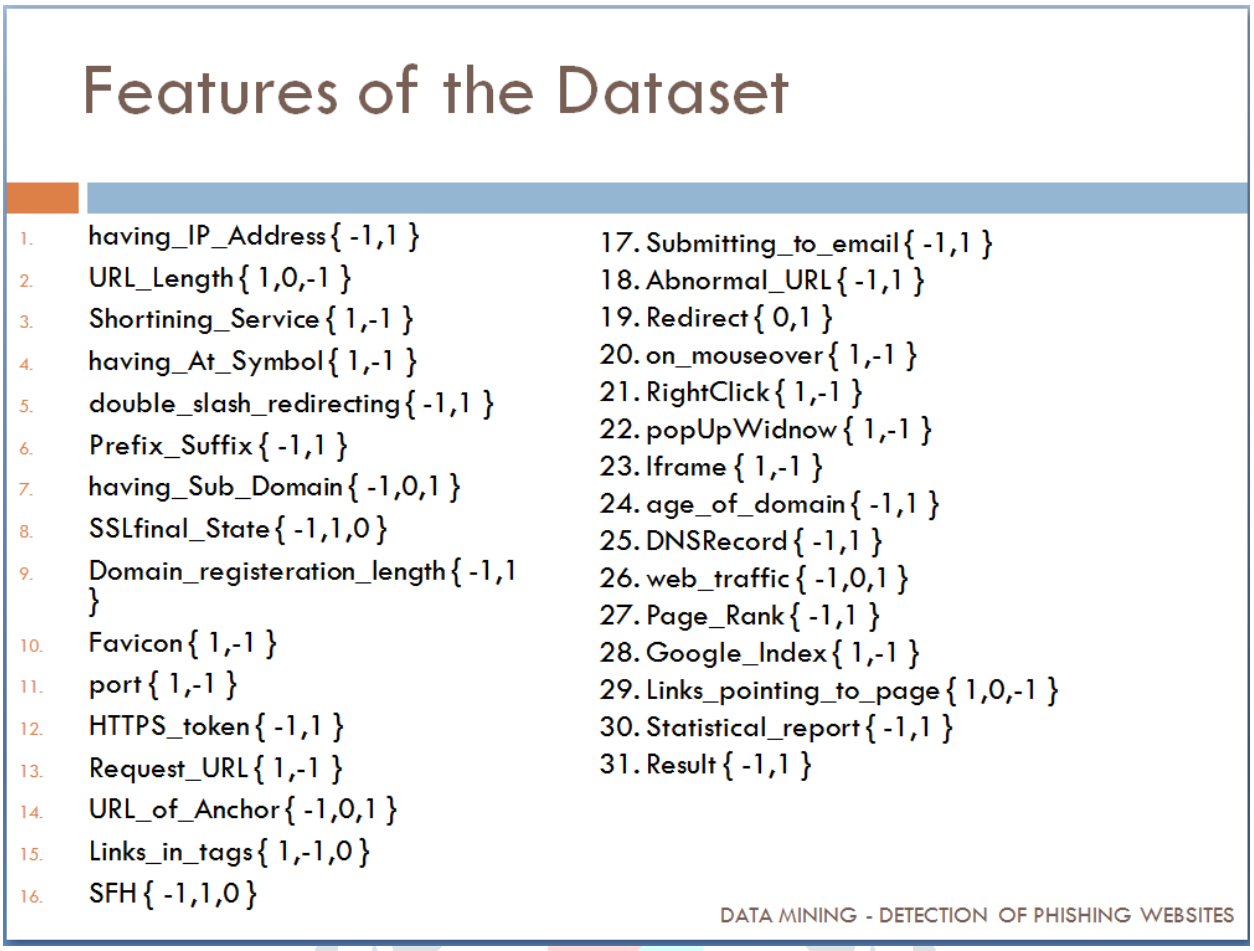
One of the challenges faced by our research was the unavailability of reliable training datasets. In fact, this challenge faces any researcher in the field. However, although plenty of articles about predicting phishing websites have been disseminated these days, no reliable training dataset has been published publicly, may be because there is no agreement in literature on the definitive features that characterize phishing web pages, hence it is difficult to shape a dataset that covers all possible features.
In this dataset, we shed light on the important features that have proved to be sound and effective in predicting phishing websites. In addition, we propose some new features.

Abstract: This dataset collected mainly from: PhishTank archive, MillerSmiles archive, Googleâ€™s searching operators.

Data Set Characteristics:	N/A	**Number of Instances:**	2456	**Area:**	Computer Security
Attribute Characteristics:	Integer	**Number of Attributes:**	30	**Date Donated**	2015-03-26
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	70266

The dataset was collected by analyzing a collection of `2456` websites among which some were used for phishing and others not. For each website included in the dataset, `30` attributes are given. The list includes for instance the URL length, whether the website uses pop-up windows or iframes, or how old the domain registration is.

The “result” feature is labeled by `-1` if it is not a phishing website and by `1` if it is a website used for phishing.



4. Packages Used

4.1. For Python

The Scikit-learn package: http://scikit-learn.org/stable/

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license
- It has been installed using pip

NumPy:

NumPy is the fundamental package needed for scientific computing with Python. This package contains:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- basic linear algebra functions
- basic Fourier transforms
- sophisticated random number capabilities
- tools for integrating Fortran code
- tools for integrating C/C++ code

Besides its obvious scientific uses, *NumPy* can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined. This allows *NumPy* to seamlessly and speedily integrate with a wide variety of databases.

NumPy is a successor for two earlier scientific Python libraries: NumPy derives from the old *Numeric* code base and can be used as a replacement for *Numeric*. It also adds the features introduced by *Numarray* and can also be used to replace *Numarray*.

FOR R:

CARET

The caret package (short for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation

As well as other functionalities.

There are many different modeling functions in R. Some have different syntax for model training and/or prediction. The package started off as a way to provide a uniform interface the functions themselves, as well as a way to standardize common tasks (such parameter tuning and variable importance).

The current release version can be found on CRAN and the project is hosted on GitHub.

doMC

The doMC package is a "parallel backend" for the foreach package. It provides a mechanism needed to execute foreach loops in parallel. The foreach package must be used in conjunction with a package such as doMC in order to execute code in parallel. The user must register a parallel backend to use; otherwise foreach will execute tasks sequentially, even when the operator is used. The doMC package acts as an interface between foreach and the multicore functionality of the parallel package, originally written by Simon Urbanek and incorporated into parallel for R 2.14.0. The multicore functionality currently only works with operating systems that support the fork system call (which means that Windows isn't supported). Also, multicore only runs tasks on a single computer, not a cluster of computers. That means that it is pointless to use doMC and multicore on a machine with only one processor with a single core. To get a speed improvement, it must run on a machine with multiple processors, multiple cores, or both.

PYTHON CODE:

```
from sklearn import tree

from sklearn.metrics import accuracy_score

import numpy as np
```

```
def load_data():

      # Load the training data from the CSV file training_data =

      np.genfromtxt('dataset.csv',
delimiter=',', dtype=np.int32)

      # Extract the inputs from the training data array (all columns but the last
one)
      inputs = training_data[:,:-1]

      # Extract the outputs from the training data array (last column)

      outputs = training_data[:, -1]

      # Separate the training (first 2,000 websites) and testing data (last
456)

      training_inputs = inputs[:2000] training_outputs

      = outputs[:2000] testing_inputs = inputs[2000:]

      testing_outputs = outputs[2000:]

      # Return the four arrays

      return training_inputs, training_outputs, testing_inputs,
testing_outputs



if___name_____== '_main__':

      print "Tutorial: Training a decision tree to detect phishing websites"

      # Load the training data

      train_inputs, train_outputs, test_inputs, test_outputs =
      load_data()

      print "Training data loaded."

      # Create a decision tree classifier model using scikit-learn

      classifier = tree.DecisionTreeClassifier() print "Decision

      tree classifier created."

      print "Beginning model training."

      # Train the decision tree classifier classifier.fit(train_inputs,

      train_outputs) print "Model training completed."

      # Use the trained classifier to make predictions on the test data

      predictions = classifier.predict(test_inputs) print predictions

      print "Predictions on testing data computed."

      # Print the accuracy (percentage of phishing websites correctly
      predicted)

      accuracy = 100.0 * accuracy_score(test_outputs, predictions)
```

```
print "The accuracy of your decision tree on testing data is: " +
str(accuracy)
```

Explanation:

- A decision tree is created by using the scikit-learn library in Python and has been used for the further components of this project.
- This will first train the decision tree on `2,000` websites, then use the trained model to predict whether `456` websites are used for phishing or not (these websites were not analyzed during training). The threshold for accuracy of the model on the testing data is 90%.

R SCRIPT:

```
library(caret)

library(doMC)

# Register 4 cores for parallel computing registerDoMC(4)

# Read the data from csv file

data <- read.csv('Datasets/phising.csv', header = F, colClasses = "factor")


# Names list for the features

names <- c("has_ip", "long_url", "short_service", "has_at",
"double_slash_redirect", "pref_suf", "has_sub_domain", "ssl_state",
"long_domain", "favicon", "port", "https_token", "req_url",
"url_of_anchor", "tag_links", "SFH", "submit_to_email", "abnormal_url",
"redirect", "mouseover", "right_click", "popup", "iframe", "domain_Age",
"dns_record", "traffic", "page_rank", "google_index", "links_to_page",
"stats_report", "target")


# Add column names names(data)

<- names

# Set a random seed so we can reproduce the results set.seed(1234)

# Create training and testing partitions

train_in <- createDataPartition(y = data$target, p = 0.75, list = FALSE)

training <- data[train_in,] testing <-

data[-train_in,]

###################### Boosted Logistic Regression
########################


# trainControl for Boosted Logisitic Regression

fitControl <- trainControl(method = 'repeatedcv', repeats = 5, number =
5, verboseIter = T)

# Run a Boosted logisitic regression over the training set
```

```
log.fit <- train(target ~ .,              data = training, method =
"LogitBoost", trControl = fitControl, tuneLength = 5)


# Predict the testing target

log.predict <- predict(log.fit, testing[,-31])

confusionMatrix(log.predict, testing$target)


###################### TreeBag
########################


# trainControl for Treebag

fitControl = trainControl(method = "repeatedcv", repeats = 5, number =
5, verboseIter = T)

# Run a Treebag classification over the training set

treebag.fit <- train(target ~ ., data = training, method = "treebag",
importance = T)

# Predict the testing target

treebag.predict <- predict(treebag.fit, testing[,-31])


confusionMatrix(treebag.predict, testing$target)


###################### Random Forest
########################


# trainControl for Random Forest

fitControl = trainControl(method = "repeatedcv", repeats = 5, number =
5, verboseIter = T)


# Run a Treebag classification over the training set

rf.fit <- train(target ~ ., data = training, method = "rf", importance =

T, trControl = fitControl, tuneLength = 5)

# Predict the testing target

rf.predict <- predict(rf.fit, testing[,-31])

confusionMatrix(rf.predict, testing$target)

plot(varImp(rf.fit))
```

Explanation:

The code first prepares the data according to the testing and training sets. Then we

have created three models for fitting and classification of the data First is Boosted

Logisitic Regression:

To our understanding, boosting is a way to convert a set of weak learners to a strong model. The weak learners specialize on different subsets of data. For example, you can iteratively build weak models. The subsequent models will do the classification task on the

misclassified data. The final model can be a weighted sum of your weak models. With boosting, you can get better results since it can reduce bias as well as variance.

Second is TreeBag Model:

Bootstrap aggregation or Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid over fitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

In the treebag model, what happens is instead of classifiers, decision trees are created and then bagging between different decision trees takes place to create the final structure of the classifier with high accuracy.

Third is Random Forest Model:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision tree's habit of over fitting to their training set.

The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, and "Random Forests" is their trademark. The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

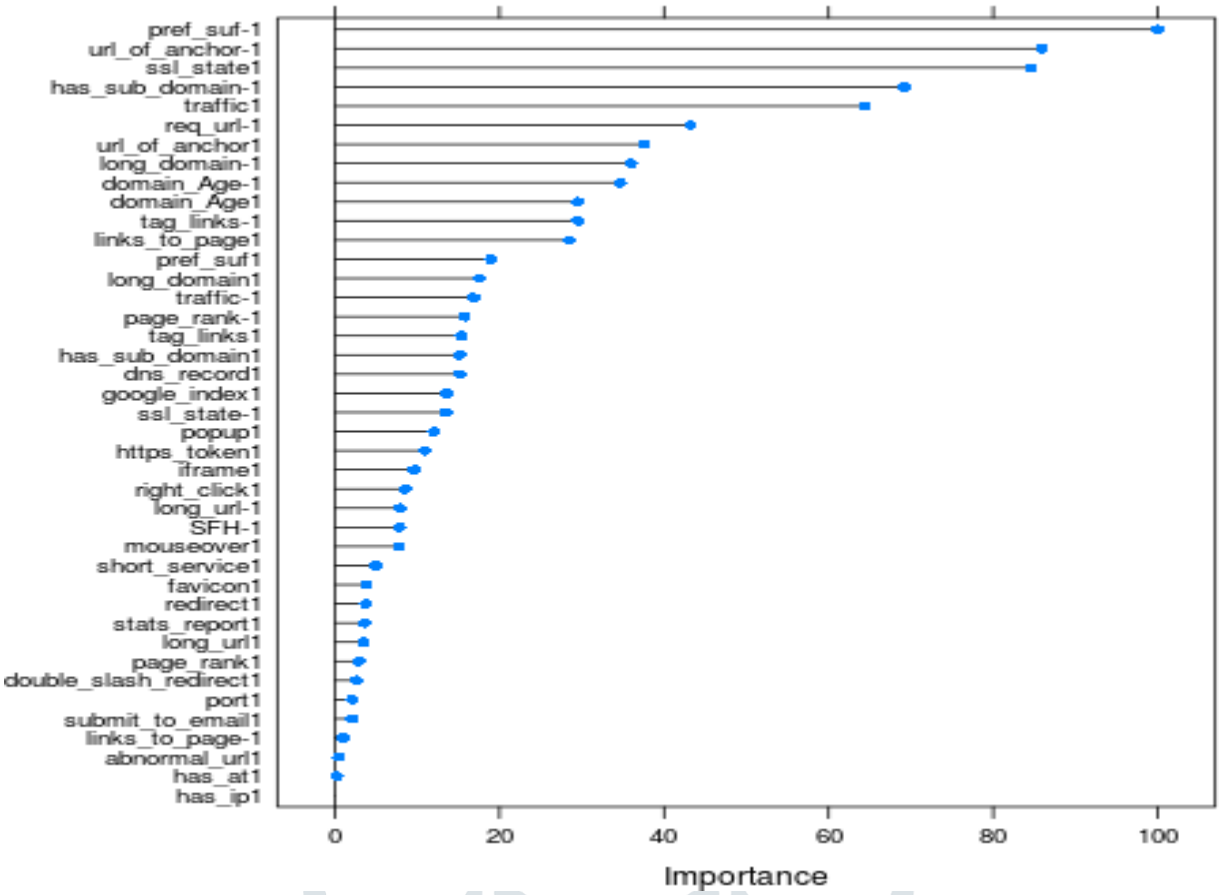
5. Results

PYTHON OUTPUT:

```
dranzer@dranzer:~/Desktop/dopudm/phishing-detection$ python decision_tree.py
Tutorial: Training a decision tree to detect phishing websites
Training data loaded.
Decision tree classifier created.
Beginning model training.
Model training completed.
Predictions on testing data computed.
The accuracy of your decision tree on testing data is: 90.6902263943
dranzer@dranzer:~/Desktop/dopudm/phishing-detection$
```

R O/P:

```
Aggregating results
Selecting tuning parameters
Fitting mtry = 11 on full training set
Confusion Matrix and Statistics

          Reference
Prediction  -1   1
        -1 331   9
        1    9 264

               Accuracy : 0.9706
                 95% CI : (0.954, 0.9825)
    No Information Rate : 0.5546
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9406
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9735
            Specificity : 0.9670
         Pos Pred Value : 0.9735
         Neg Pred Value : 0.9670
             Prevalence : 0.5546
         Detection Rate : 0.5400
   Detection Prevalence : 0.5546
      Balanced Accuracy : 0.9703

       'Positive' Class : -1
```



6. Conclusion:

For the Python Code:

We have performed phishing detection using Python and identified and trained models based upon the phishing websites database.

From the above code and output, we have got 90.5% accuracy in the python DecisionTreeClassifier Function which meets our threshold value of 90%

For the R code:

Received Output: 97.6% meets the threshold of 90%

We are using a 'TreeBag' model to learn from the features in the training dataset. Here, we have an extensive set of features available in the dataset, ranging from simple attributes like 'Domain Length' to more complex ones like 'Port being used', etc. we were able to get an accuracy of 0.964 with a treebag model from Caret package. The dataset is having approximately 2500instances of 30features, out of which some 1900 instances are used for training purposes. This is quite a good ratio for using a tree-based bagging model. With enough instances we can be pretty sure that model won't overfit the training data and as apparent from the results it doesn't, resulting in an accuracy of more than 96%. An ensemble tree-based model works really good in this scenario and after having a look at the variable importances of different attributes we can conclude along with our real-world experience that the most important variables found by the treebag model are indeed the most significant ones while identifying a phishing website.

Here are 5 most important attributes as found by, R Script (treebag):

1. (Abnormal URL Anchor)url_of_anchor: 100.000
2. (SSL State) ssl_state: 97.690
3. (Traffic) traffic: 76.097
4. (Prefix Suffix) pref_suf: 62.365
5. (Domain Age) domain_Age: 9.584

7. References

1) Moore, T., & Clayton, R. (2007, June). An Empirical Analysis of the Current State of Phishing Attack and Defense. In *WEIS*.

2) Florencio, D., & Herley, C. (2005). Stopping a phishing attack, even when the victims ignore warnings. *Microsoft Research MSR-TR-2005*, *142*.

3) B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. A browser plug-in solution to the unique password problem. To appear in Proceedings of the 14th Usenix Security Symposium, 2005.

4) Phishing Websites Predictions, this project uses Phishing Websites dataset from UCI machine learning Datasets. The objective is to identify whether a website is a Phishing website one or not.