[Automating Phishing URL Detection with Random Forest Classifiers]

[William Tiozzo, Jerry Luo, Yeji Ju]

Target

Validate phishing websites through features of URL, can be used for both enterprise and normal end users to check for the credibility of their sources. Crucial for businesses to determine a valid source of partnership.

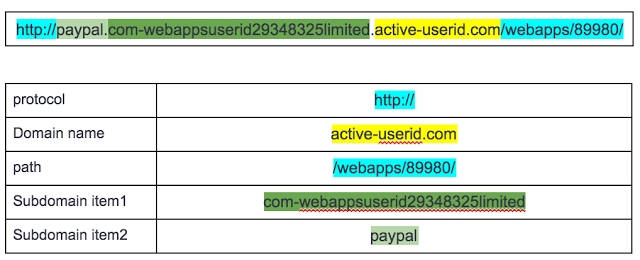
Model: Random Forest Classifier

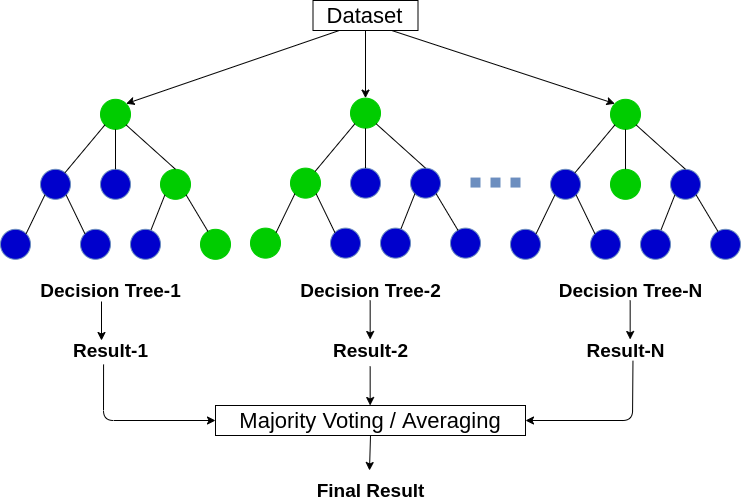
Random forest can be used for both classification and regression problems. Random forest classifiers work by constructing a variable number of decision trees that output the decisions of all decision trees into one single output. A random forest classifier is essentially a collection of decision tree classifiers that each do their best to offer a single output. The final output is determined by the “winner” class which is the one that appears most times in the list of outputs from all decision trees used.

Why use Random Forest Classifier?

This decision was based on accuracy scores of various other similar projects. Out of all the projects, random forest classifiers showed a greater advantage in terms of accuracy scores, most of which are above 80 percent for both train accuracy and test accuracy. The implementation was also easier for random forest classifiers. Another possible solution was a vector support machine as our second choice as it also supports non-linear relationships.

How can random forest classifiers classify phishing websites?

One example is through words or text, the classifier pulls individual words from the url and determines whether they belong to the typical wordlist used in phishing websites. The random forest model comprises a set of decision trees each using random subsets of features such as text from the url. 



(Random forests create decision trees on randomly selected data samples, final output is determined by winner class)

Advantages:

- Reduces overfitting in decision trees and helps to improve accuracy

- Normalizing data is not required as it uses rule-based approach

- Flexible to both classification and regression

Disadvantages:

- Requires more computational power as well as resources as it builds numerous trees to combine multiple outputs

- Due to an ensemble of decision trees, it also suffers interpretability and fails to determine significance of each variable

- Can require much time for training as it combines decision trees to determine the class

Dataset used for training:

Dataset1: <https://www.kaggle.com/eswarchandt/phishing-website-detector>

- A collection of website URLs for 11000 + websites. Each sample has 30 website parameters and a class label identifying it as phishing website or not (1 or -1)

Dataset features: IP, LongUrl, ShortUrl, Symbols, Redirecting …

Dataset2: [https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techiniques/blob/master/DataFiles/4.phishing.csv](https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techiniques/blob/master/DataFiles/3.legitimate.csv)

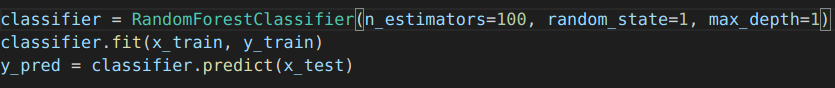
Dataset2 features: IP, URL Length, URL\_Depth, Redirection …

- Using pandas to concatenate dataset1 and dataset2 hoping for improvements in terms of accuracy scores, the accuracy score unfortunately did not improve (remained at 85%)





Training



The “n\_estimator” parameter determines the number of trees in the forest so in this case, the number of trees is set to 100. The “random\_state” parameter controls both the randomness of the bootstrapping (machine learning ensemble meta-algorithm designed to improve stability and accuracy of machine learning algorithms used in statistical classification and regression, also reduces variance and helps to avoid overfitting), random state is set to 1 to improve the randomness of sampling for the data so the output can be more representative. “max\_depth” is the maximum depth of the tree, which is set to 1, the accuracy score was around 85%.



Classification Report



TN / True Negative: Case was negative and predicted negative

TP/ True Positive: Case was positive and predicted positive

FN / False Negative: Case was positive but predicted negative SSS

FP / False Positive: Case was negative but predicted positiveS

Precision: Number of identified members of class divided by all the times the model predicted the class

Precision = TP / (TP + FP)

Recall: Number of members of class that the classifier identified correctly divided by total number of members in that class

Recall = TP / (TP + FN)

F1 Score: Combines precision and recall into one metric, if precision and recall are both high, F1 will be high too

F1 Score = 2 \* (Recall \* Precision) / (Recall + Precision)

Validation Curve for Visualization

(The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples)

Overfitting and Underfitting

Overfitting – When the model does not generalize well from the training data to unseen data. The model captures spurious patterns that won’t recur in the future leading to less accurate predictions.

Underfitting – Occurs when a model is too simple, informed by too few features or regularized too much which makes it inflexible in learning from the dataset.

How to look at the validation curve?



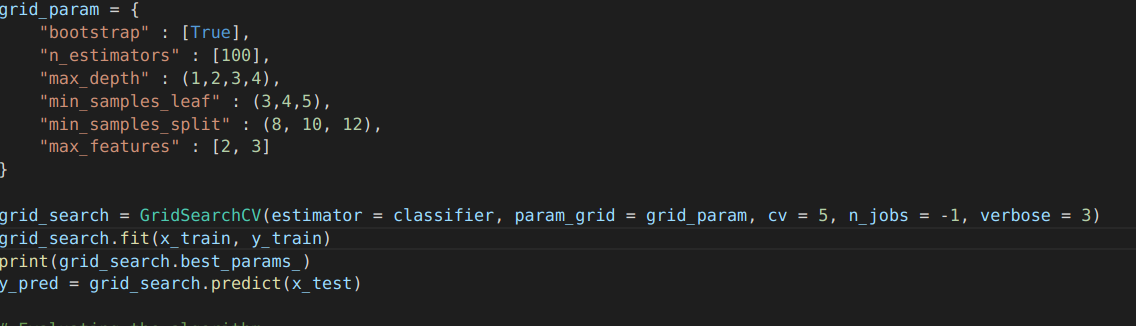
Overfitting – Occurs when the training curve reaches a high score relatively quickly and the validation curve is lagging behind. This means the model is very complex and there is too little data.

Underfitting – If both validation and training scores are low, the model is too simple or it is informed by too few features.

A validation score is an important diagnostic tool that shows the sensitivity between changes in a machine learning model’s accuracy with change in parameters of the model. If the training curve reaches a high score quickly and the validation curve is lagging behind, the model is overfitting. This means the model is very complex and there is too little data which is what's happening to our model. The validation score is below the training score which means the model is overfitting.

Optimizing Accuracy with GridSearchCV

GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given value. There is no way to know the optimal values for the model so it is necessary to try all possible values to know the optimal parameters. Doing this manually could take a considerable amount of time and resources so using GridSearchCV to automate the tuning of hyperparameters could save a lot of time.



Result:

{'bootstrap': True, 'max\_depth': 4, 'max\_features': 3, 'min\_samples\_leaf': 4, 'min\_samples\_split': 8, 'n\_estimators': 100}

After trying more than 100 types of combinations for our parameters, GridSearchCv determined that the above was the best parameters for our model. Bootstrap method is used to estimate the skill of machine learning models when making predictions on data not included in training data; by default, bootstrap is set to true for random forest classifiers. The “max\_feature” parameter is to set a maximum number of features a random forest is allowed to try in each individual tree. “Min\_samples \_leaf” and “min\_sample\_split” have similar ideas behind, both of which are used to specify a minimum number of samples required to decide a node to be leaf or split further. Finally, “n\_estimators” is used to set the number of decision trees best for the model and training data.

Next Steps

* Construct GUI by creating browser extension
* Specific analysis on why URL is malicious, pi chart, bar graph …
* Mass data collection for malicious URL, construct on database for malicious URL and update to the model

Summary

Validating malicious URLs with random forest classifiers resulted in an accuracy of around 78%, optimization was required to increase accuracy. For optimization, hyper parameterization was the first option, increasing one parameter (max\_depth) allowed the accuracy to increase to 85%. In order to maximize our accuracy, another option to further increase accuracy score was by using gridsearchcv which unfortunately did not help. Still, random forest classifier proved to be a great model for URL validation with a finalized accuracy of 85%.

References

[Validation Curve Explained — Plot the influence of a single hyperparameter | by Rukshan Pramoditha | Towards Data Science](https://towardsdatascience.com/validation-curve-explained-plot-the-influence-of-a-single-hyperparameter-1ac4864deaf8)

[Random Forest Classifier and its Hyperparameters | by Ankit Chauhan | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6)

[Optimizing Hyperparameters in Random Forest Classification | by Reilly Meinert | Towards Data Science](https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6)

[sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

[Simple Text Classification using Random Forest. | by Tenzin Ngodup | Medium](https://medium.com/@tenzin_ngodup/simple-text-classification-using-random-forest-fe230be1e857)

For the structure of a research paper, it includes these parts

Abstract

Introduction

Experimentation

Results and Discussions

Conclusions

Reference