Phishing and Malicious URL Detection

Hongbo Shen: hs3224 **Junfeng Wu: jw4158**

Yuqing Guan: yg2756 **Jiajun Han: jh4316**

Section 1. Introduction

Today people live in a rapidly changing information world. However, in this information society, personal information becomes vulnerable under some hackers' attacks. One of the common types of cyber-attack is called Phishing which often occurs on webs, mobile applications and emails.

One good thing for us is that most Phishing can be identified successfully with machine learning methods. For example, the most common phishing method is *Link Manipulation*. Cybercriminals use different methods to fake a URL link to make it resemble one legitimate URL. For example, malicious URLs are represented as hyperlinks with names on websites, misspelled URLS are deliberately used to make it look like a legitimate URL, and non-English characters are covertly replaced but look like English characters to visually confuse web users. Besides, there are many other phishing methods besides link manipulation. *Filters Circumvent* showing their site content in figures or using flash, making it hard to detect by phishing detection methods. *Site Forgery* occurs when a legitimate site manipulates the target site's JavaScript code. As for our project, we mainly work on the way to detect link manipulation.

In this paper, instead of focusing on the lower-level cyber principles of these phishing methods, we will address this problem from a statistical view, extracting the common difference of features between the phishing mail and normal ones, and

generalizing the features in a model by a range of different algorithms. In the experiment, we applied and compared eight different machine learning methods: Logistic Regression, Decision Tree, KNN, SVM, Random Forest, Adaboost, Gradient Boosting and XGBoost. We trained and tested these models with the *Website Phishing Data Set* (https://archive.ics.uci.edu/ml/datasets/Website+Phishing) which is collected from Phishtank Data Archive (www.phishtank.com).

More details of the datasets and reference papers will be discussed in Section 2.

Section 3 first introduces the detailed basic principles of the eight machine learning methods mentioned above and our reproduction of the algorithms. Then, we will discuss the difference of experiment results between this paper and the reference papers, and provide an illustrative overview.

In Section 4 we will introduce new methods that have not been mentioned or implemented in the reference papers.

Finally, Section 5 is the conclusion of the paper and some expectations for future works.

Section 2. Dataset and Papers

In this part we introduce more details about the Website Phishing Data Set. This dataset contains 1353 samples and 10 dimensions, including the dependent variable "*Result*" column. The 10 dimensions are *SFH, web_traffic, Request_URL, popUpWindow, SSLfinal_State, URL_of_Anchor, URL_Length, age_of_domain, having_IP_Address,* and *Result*, where *Result* is a dependent variable and others are independent variables. Some reference papers applied a type of more informative 30 dimensions dataset which includes all 10 dimensions mentioned above and with some

extra variables such as *DNS Record, Double Slash Redirection,* and *HTTPS token*. However, we will only discuss the phishing relationship among the 10-dimension dataset. The features of the dataset are as follows, all variables are 3-classed with 1 represents for legitimate, -1 represents for phishing and 0 represents for suspicion:

(1) SFH: *SFH* is short for Server From Handler. In this dataset, this variable is labeled as 0 if the domain name in *SFHs* is different from the domain name of the webpage, or denoted as -1 if *SFH is 'about : blank' or empty.*

(2) popUpWindow: Indicates having pop-up windows on the webpage. It is denoted by -1 if *right click disabled*, by 0 if *right click showing alert*.

(3) SSLfinal_State: Shows if the website uses SSL or not and whether HTTPs protocol is issued by a trusted vendor. Labeled as 1 if *use of https & trusted issuer & age > 2 years*, as 0 if *using https & issuer is not trusted*, as -1 for else.

(4) Request_URL: Indicates whether it contains external objects from other servers in a single web page. Denoted by 1 if URL < 22%, by 0 if request URL>22% and < 61%, by -1 for else.

(5) URL_of_Anchor: Anchor means the < a > tag. Indicates whether it points to a different domain, which is not the one typed in the URL address bar. Labeled as 1 if *URL anchor < 31%, as 0 if URL anchor > 37% and < 67%*, as -1 for else.

(6) web_traffic: It measures the popularity of the website by its rank in the Alexa Database. Labeled as 1 if *webTraffic < 150000*, as 0 if Legit *webTraffic > 150000*.

(7) URL_Length: Whether it has a long URL because sometimes long URL can

be used as the camouflage for phishing in the address bar. Labeled as 1 if *URL length < 54*, as 0 if *URL length > 54 and < 75*, otherwise as -1.

(8) age_of_domain: If the age of the domain is less than a certain age. Labeled as -1 if *age < 6 months*, otherwise as 0.

(9) having_IP_Address: If there is an IP address instead of the domain name in the URL. Labeled as -1 if *IP address exists in URL*, as 1 otherwise.

(10) Result: "1" represents Legitimate and "0" denotes either Phishing or Suspicious (means probably Phishing so we cannot ignore it and handle it as Phishing).

However, in the practical application, for instance, in the automatic spam mail recognition system, the classifier usually has only two choices: show the mail to the account user (labeled as not spam) or through it into the spam box (labeled as spam). Therefore, we process the dataset and see 0 and -1 as the same phishing, then simplify the classification problem into binary. We can make evaluations for more metrics, such as *accuracy, precision, recall* and *F1-score*; these metrics will be introduced in more detail in Section 3 along with experiment results. Before starting the experiment, let's first analyze some basic properties of the dataset. The correlation matrix of the dataset is shown as the figure below.

Observing the matrix, Result is much more strongly correlated with *URL_Anchor than other variables,* and *popUpWindow* seems weakly correlated with all variables. Maybe a bunch of statistics are not clear enough to get much information at the first

glance, a visualized plot form is shown as the figure below.

Observing the correlation plot, among all correlation pairs, except for *Result* and *SSL_final_State, Result and URL_of_Anchor* are strongly positively correlated, all other combinations tend to be uncorrelated. Overall speaking, all features tend to be uncorrelated with each other.

```
> cor(dataset)
                        SFH   popUpWidnow SSLfinal_State  Request_URL URL_of_Anchor
SFH              1.000000000 -4.862726e-03     0.17140239  0.126660605    0.11431133
popUpWidnow     -0.004862726  1.000000e+00    -0.01300489 -0.004622095    0.04115019
SSLfinal_State   0.171402389 -1.300489e-02     1.00000000  0.193054440    0.53578619
Request_URL      0.126660605 -4.622095e-03     0.19305444  1.000000000    0.17769320
URL_of_Anchor    0.114311328  4.115019e-02     0.53578619  0.177693201    1.00000000
web_traffic      0.052706363 -4.319003e-02     0.25876784  0.161166301    0.32629323
URLURL_Length    0.414196219 -4.938124e-02     0.04875373  0.246348096   -0.02339551
age_of_domain   -0.015839916 -9.482202e-04     0.16280942  0.090455473    0.07550813
having_IP_Address -0.010962287  9.688229e-02     0.07141450  0.029772867    0.09984696
Result           0.221419008  8.588679e-05     0.71474120  0.253372272    0.69293452
                 web_traffic URLURL_Length age_of_domain having_IP_Address       Result
SFH              0.052706363   0.414196219 -0.0158399160      -0.010962287 2.214190e-01
popUpWidnow     -0.043190028  -0.049381235 -0.0009482202       0.096882295 8.588679e-05
SSLfinal_State   0.258767835   0.048753733  0.1628094196       0.071414500 7.147412e-01
Request_URL      0.161166301   0.246348096  0.0904554734       0.029772867 2.533723e-01
URL_of_Anchor    0.326293230  -0.023395510  0.0755081304       0.099846955 6.929345e-01
web_traffic      1.000000000   0.008992782  0.0899489497       0.002922205 3.461031e-01
URLURL_Length    0.008992782   1.000000000  0.1794264244      -0.052410739 5.742963e-02
age_of_domain    0.089948950   0.179426424  1.0000000000      -0.010445721 1.214964e-01
having_IP_Address 0.002922205  -0.052410739 -0.0104457207       1.000000000 9.416009e-02
Result           0.346103108   0.057429629  0.1214964165       0.094160095 1.000000e+00
```

Fig 1. Correlation matrix of the dataset

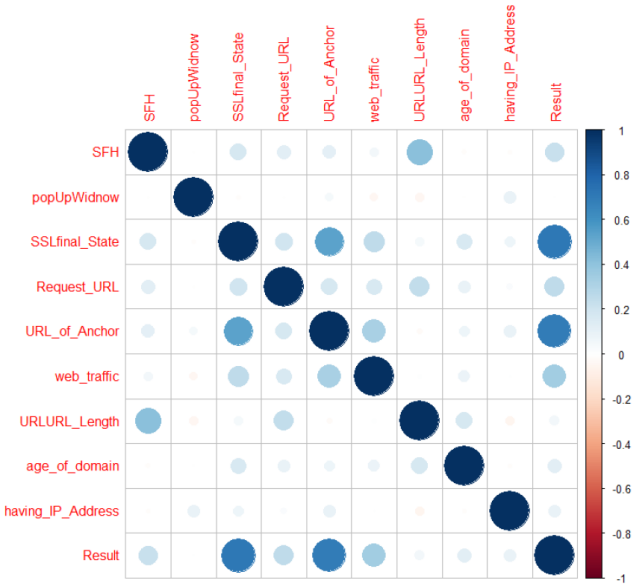

Fig 2. Correlation plot of the dataset

Section 3. Experiment Results Comparison

In this project, we will apply eight machine learning approaches to classify the phishing dataset through R language.

One good thing for us is that most Phishing can be identified successfully with machine learning methods. It has made considerable contributions in real time classification problems. Machine learning models are adaptive to a very wide range of identification patterns of cyber malicious actions. Therefore, machine learning is a very powerful tool of phishing detection. So far, there are various machine learning methods used to evade common phishing attacks, just like in Section 2 we mentioned the reference papers and we will briefly introduce and recover a part of the machine learning methods implemented in the reference papers. After that, we will evaluate and list out the experiment results in a variety of metrics and make a simple comparison and come up with some concise conclusions.

I. Logistic Regression

Before introducing the implementation of Logistic Regression, let me first introduce the sigmoid function. Sigmoid function is simple:

$$S(x) = \frac{1}{1+e^{-x}}$$

The advantage of sigmoid function is to map a possible infinite number into the range of (0,1). There are two major meanings. First, this prevents the risk of value overflow in the computational simulation. Besides Ridge Regression, as a binary classifier, the sigmoid value between 0 and 1 clearly and straightforwardly shows the probability, or say, the confidence of which one of positive or negative this sample should belong to.

In this experiment, we use Logistic Regression to train binary (K=2) classifications. It first transforms the output by sigmoid function to a probability value that can then be mapped to negative or positive.

II. KNN

KNN is supervised learning and it uses the similarity between data, and puts the data point into the class in which they have the most similarities. It can be used for both regression and classification. It is lazy and non-parametric because it is unnecessary to train the KNN model, it simply classifies the data point for the given dataset by putting the data point into the most similar category.

III. SVM

SVM belongs to a supervised learning algorithm and can be used for both classification or regression. The main idea of SVM is separating points using surfaces. Similar to other supervised learning, SVM requires time to train models and its training dataset is classified at different points in space, which are classified into different groups using different kernels. The objective of this algorithm is to find out the optimal separation of data, that is the Maximal margin hyperplane which maximizes the boundary between the two data categories.

IV. Decision Tree

Decision tree classifiers is a tree structure algorithm. The internal node denotes one attribute, the branch identifies the corresponding decision rule, and leaf node represents the outcome. The root node partitions the tree in a recursive manner based on the attribute value, which is called recursive partitioning. Therefore, the decision

tree can deal with numerical and categorical data sets.

Also, decision trees are suitable to deal with nonlinear relationships between features. Gini Index is often used as an impurity function to evaluate the quality of each partition.

V. Random Forest

Random Forest trains a certain number of trees and enables the model to make decisions. Each tree in a random forest specifies the class prediction in classification tasks, and the mean prediction for regression tasks. The result will be the most predicted class. Random forest has the ability to rectify itself, reduces overfitting and bias, and estimates the lost data. In general, random forests perform better than decision trees. However, since random forest introduces randomness to the training and the testing datasets and has independence between decision trees, it lacks reproducibility and interpretability.

VI. AdaBoost

AdaBoost is a branch of Boosting methods, which is short for “Adaptive Boosting”. The principle of its adaptive is that, the weight increases when the former classifier makes a mistake, and decreases when it is right, and then trains the next classifier. Meanwhile, in every iteration, introduce a new weak classifier, until the algorithm reaches a small error rate or max iteration. In short, AdaBoost will iterate over and over again to select train dataset based on the previous training outcomes, and can be used in classification with high prediction accuracy.

VII. Gradient Boosting

Gradient boosting is developed based on AdaBoosting, and it's also called gradient tree boosting. Different from random forest where each tree is independently trained, boosting trains each tree iteratively. There are three main components in Gradient boosting : the loss function, the weak learners, and an additive model.

In gradient boosting, there is no need to derive a new boosting algorithm for every loss function used, so any type of loss function can be used. Thus, gradient boosting is also more robust to outliers compared with AdaBoost. A gradient descent method can be used to minimize the loss function, thus, each tree added to the model reduces the loss. Once the loss reaches an acceptable number, the training process will stop.

VIII. XGBoost

XGBoost, which is the abbreviation of "Extreme Gradient Boosting", is one scalable machine learning algorithm for boosting, providing better performance and speed compared with Gradient Boosting.

The most significant thing about XGBoost is its high scalability, as a result of algorithmic optimizations, because parallel and distributed computing make XGboost learn very fast and enable fast model exploration. Besides, XGBoost uses out-of-core computation so that hundreds of millions of data can be processed on a desktop.

IX. Model Evaluation and Comparison

The results of different methods are displayed in the following table. One thing to be mentioned is that when applying 4 SVM variable sets in this experiment, the

format description is too long to be written into the table. SVM I stands for SVM (linear, cost=5), SVM II stands for SVM (RBF, cost=1, gamma=0.5), SVM III stands for SVM (polynomial, cost=100, degree=3), SVM IV stands for SVM (sigmoid, cost=0.1, gamma=0.5).

	train time(s)	test time (s)	accuracy	recall	precision	F1 score
Logistic Regression	2.4285390	0.2324009	0.9064039	0.8975904	0.8764706	0.886904
KNN(k=5)	0.03600192 (requires no training)		0.91133	0.9036145	0.8823529	0.8373494
SVM I	0.07701302	0.01100707	0.9014778	0.9096386	0.8579545	0.8830409
SVM II	0.1300521	0.02200198	0.91133	0.9036145	0.8823529	0.8928571
SVM III	0.122999	0.02000213	0.9162562	0.9216867	0.8793103	0.9
SVM IV	0.07303214	0.01199985	0.7438424	0.6445783	0.7039474	0.672956
Decision Tree	0.003000021	0.004001141	0.8768473	0.8915663	0.8222222	0.855491
Random Forest	0.1879148	0.00782514	0.92840	0.9085366	0.9141104	0.863221
AdaBoost	0.03291202	0.00598383	0.9039409	0.8915663	0.8757396	0.883582
Gradient Boosting	4.580734	0.00646186	0.89630	0.9146341	0.8658537	0.877193
XGBoost	0.03294182	0.00400782	0.9039408	0.8915662	0.8757396	0.883582

Observing the experiment statistics, most algorithms have an accuracy slightly greater than 0.9 and a F1 score slightly lower than 0.9. Logistic Regression and

Gradient Boosting uses much more time than other algorithms, this is because these two algorithms update a new tree in each training epoch. However, the accuracy and F1 score no obvious difference with other algorithms. Compared to the four different kernels of SVM, SVM III is the best and SVM IV is the worst, one reaching a F1 score of 0.9 and another only less than 0.68.

To find the best performance of SVM, we have tried four different kernels: Linear, Polynomial, Sigmoid and Radial Basis Function (RBF) Kernel. Parameters of each kernel (like cost, gamma and degree) can be optimized with a cross-validation model selection. In our project, we use the function *tune()* in library 'e1071' to choose the parameters. However, with more than one hyperparameter to tune, automatic model selection is prone to overfitting. As we can see below, SVM with Polynomial and RBF kernel both can be considered as a good default prediction model for our problem, as we lack the expert knowledge.

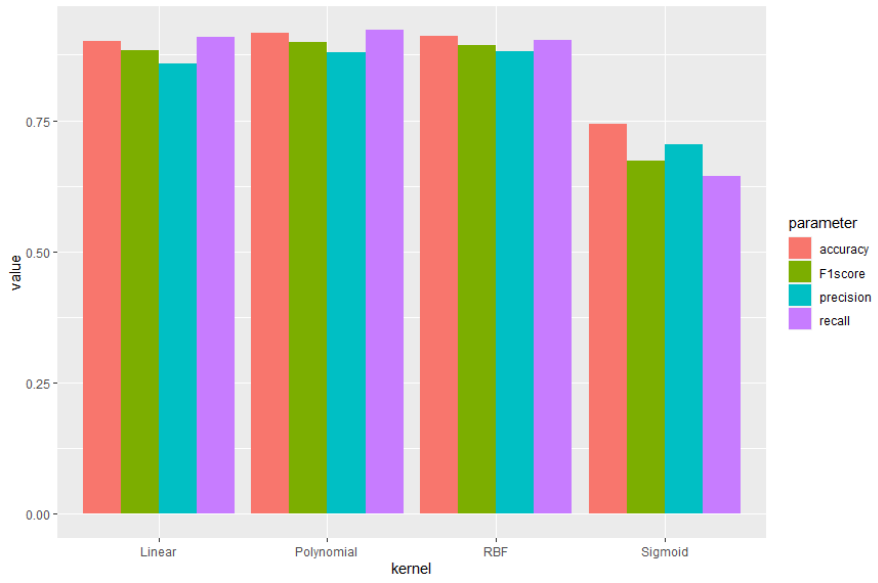

Fig 3. SVM with different kernels

As for KNN, the best model in our case is K=5 after comparing the number of neighbors from 3 to 10. If K is small, then its decision boundary will be more flexible,

that is this KNN model has low bias but relatively high variance. When K becomes larger, it will be less flexible and has a linear-like decision boundary, which means high bias but low variance. For our data set, KNN with K = 5 gives us a good prediction, as shown below:

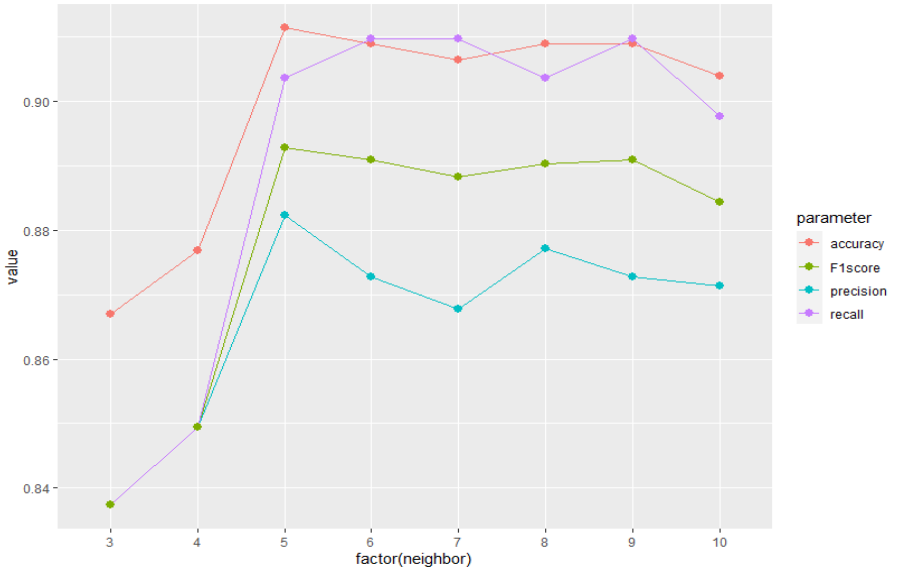

Fig 4. KNN with different number of neighbors

The main advantage of XGBoost and AdaBoost is its speed, and it's regularization variable that successfully reduces variance. However, XGBoost is harder to understand, visualize, and to tune compared to AdaBoost and Random Forests. There are a bunch of hyper-parameters that can be tuned to increase performance. In comparison, AdaBoost is easy to visualize and we can easily get a figure of the importance of every variable during the AdaBoost training.

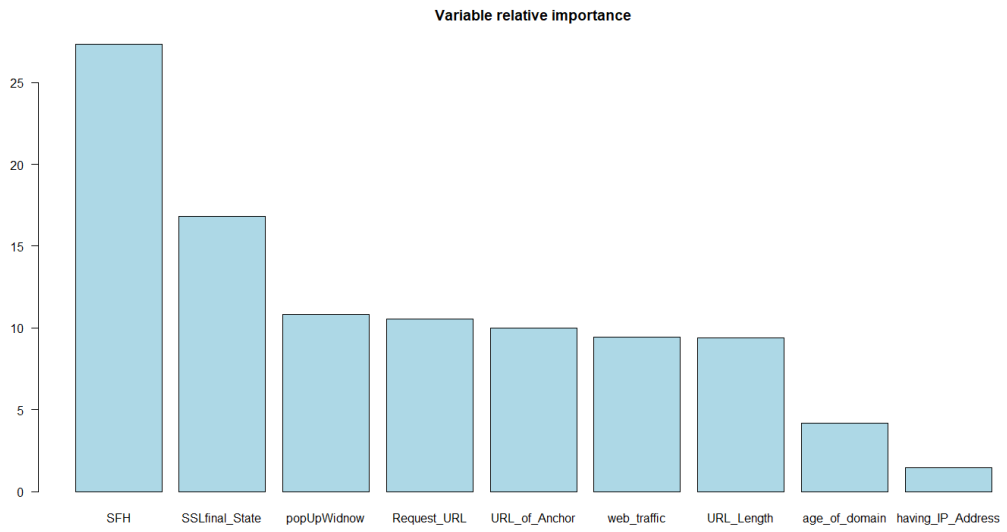

Fig 5. AdaBoost

Section 4. Introduce other methods

As we have analyzed in Section 2, the variables tend to be uncorrelated with each other in this dataset. Intuitively simple linear models can be a very easy and effective method in uncorrelated datasets. Therefore, we decided to test the performance of the linear model and make analytical comparisons between approaches mentioned in Section 3 in this experiment.

Because this project is a binary classification problem, we set the prediction threshold at 0.5 when the predicted value is distributed between 0 and 1. The performance of fitting a linear model between Result and all other variables are listed as the following table.

train time(s)	test time(s)	accuracy	recall	precision	F1 score
0.0149622	0.008975029	0.9187192	0.9497908	0.9153226	0.9322382

Observe and compare between linear model and Section 3 methods, the general performance of linear model has no obvious disadvantages, and even surpassed some algorithms, compared to the algorithms mentioned in Section 3. That is just like what we have predicted at the top of Section 4, a simple linear model is fast, and can be accurate simultaneously if the dataset variables tend to be uncorrelated with each other. However, when the co-relationship between variables is strong, either positive or negative, the accuracy metrics can be low because simple linear models are not adequate to compute these.

Upon the basis of fitting linear model between Result and all other variables, we noticed that in the dataset analyze in Section 2 that except *SSLfinal_state* and *URL_of_Anchor* are obvious strongly correlated with the *Result*, other variables might be not that correlated with the *Result*. Hence, to double-verify which variables are really relevant and which ones are just a kind of noise, and reach an optimized plan to accept some variables and reject others, we implement the best subset approach. Before reaching quantitative results, firstly do some qualitative analysis here. Call the summary of fitted full linear model, the coefficients and *p-values* are shown as figure below.

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        0.497365   0.019928  24.958  < 2e-16 ***
SFH               -0.073160   0.039128  -1.870  0.06183 .
popUpWidnow       -0.003219   0.012018  -0.268  0.78891
SSLfinal_State     0.215665   0.013285  16.233  < 2e-16 ***
Request_URL        0.015393   0.010326   1.491  0.13638
URL_of_Anchor      0.267026   0.017388  15.357  < 2e-16 ***
web_traffic        0.101974   0.013627   7.483 1.67e-13 ***
URLURL_Length      0.098249   0.037928   2.590  0.00974 **
age_of_domain      0.028188   0.010274   2.744  0.00619 **
having_IP_Address -0.009599   0.015381  -0.624  0.53273
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fig 6. p-values of each variable

Observing the p-values, we can get a shortcut anticipation of conclusion, the obvious 5 big p-values corresponds to *SSLfinal_State, URL_of_Anchor, web_traffic, URL_Length, age_of_domain* that are signed as relevant while other variables can be seen as noises. Perhaps *SFH* may be a controversial variable. Hence, we should do some penetrating quantitative processes to determine an optimized model.

There are usually 3 metrics to measure the performance of a sub-model, *Mallow's Cp, Bayesian AIC (BIC)* and *adjusted* R^2. Denote n as the number of samples and RSS satisfies

$$RSS = \Sigma(y_i - \beta x_i)^2$$

Then the formula of Mallow's Cp is

$$C_p = \frac{1}{n}(RSS + 2d\hat{\sigma})$$

Where d is the number of predictors and is an estimator for the variance of the voice, the same below.

Before introducing *BIC*, here comes Akaike Information Criteria (*AIC*) first. *AIC* was founded by Japanese statistician Hirotugu Akaike in 1974, which is built on the concept of information entropy, it puts an extra punishment on models using too

many uncorrelated variables and encourages simplified models. The formula of *AIC* satisfies

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma})$$

BIC is very similar to *AIC*, founded by Schwarz in 1978, the punishment of the complex model of *BIC* is even bigger than *AIC*. Hence, BIC can be seen as an optimized version of *AIC*. The formula of *BIC* satisfies

$$BIC = \frac{1}{n}(RSS + d\hat{\sigma}^2)$$

R^2 value, ranging between 0 and 1 describes the extent to which input variables explain output variables. In general, the larger R^2 is, the better the fitting degree is. However, adding variables brutally, relevant or not, R^2will either increase or remain unchanged. Likewise *AIC* and *BIC*, *adjusted* R^2 punishes over-complex models with many irrelevant variables. The formulas of R^2 and *adjusted* R^2 are

$$R^2 = \frac{SSR}{TSS} = 1 - \frac{RSS}{TSS}$$

$$R^2_{adjusted} = 1 - \frac{(1-R^2)(n-1)}{n-p-1}$$

One thing should notice that is *adjusted* R^2 is also between 0 and 1. Select a model with the lowest *Cp, BIC* and the highest *adjusted* R^2. The summary of sub-models and plots of the values are shown as the figure below.

From 1 to 9 of all the subset size, 6 has the best general performance evaluated

by the metrics above, which corresponds to a sub-model with *SFH, SSLfinal_State, URL_of_Anchor, web_traffic, URL_Length, age_of_domain*. Then, treat this as a new linear model and evaluate this model, we can get the performance table of the new sub-model as follows.

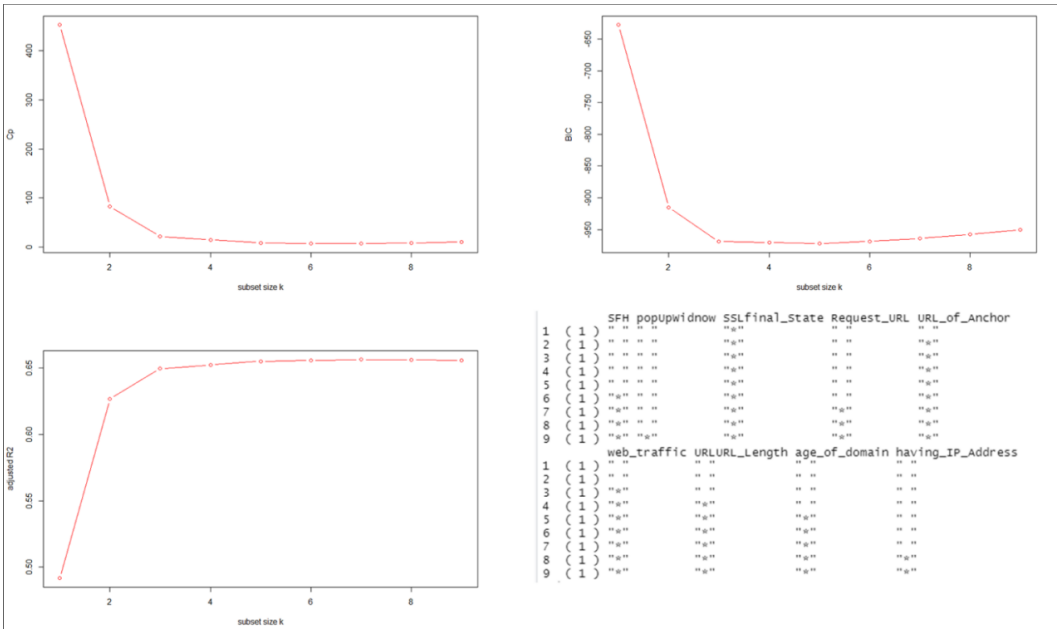

Fig 7. Cp, BIC, Adjusted R^2 and summary

train time(s)	test time(s)	accuracy	recall	precision	F1 Score
0.008957148	0.003988981	0.9211823	0.9497908	0.9190283	0.9341564

Observing the result, the new sub-model reached an even higher accuracy in a shorter time than the old full model. Therefore, in classification problems, sometimes it is important to consider the relationship between dependent variables and each independent variable.

In Section 3, we implement the decision tree method without pruning.

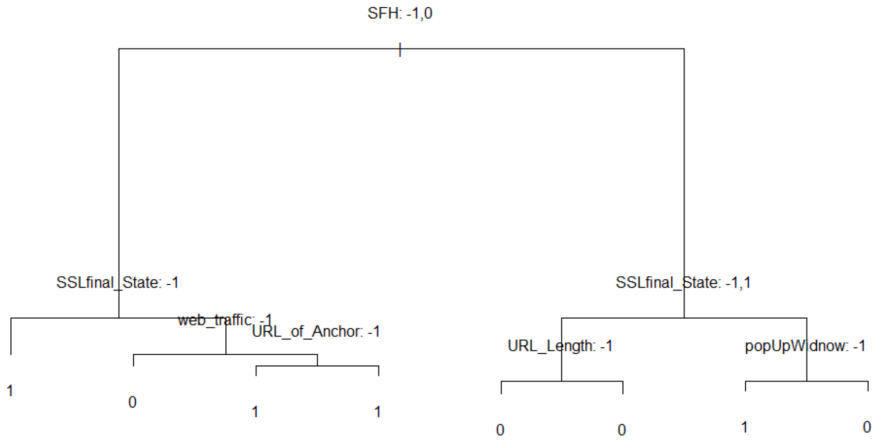

Fig 8. result of decision tree without pruning

Next, we will explore whether pruning the tree might lead to a refined result. First, we use cross-validation to determine the optimal level of tree complexity and the cost complexity pruning is used in order to select a sequence of trees for consideration.

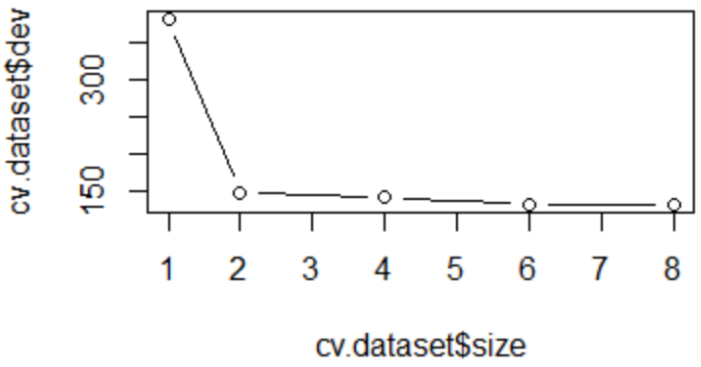

Fig 9. relationship between cross-validation error with the size of tree

As we can see, the tree with 6 and 8 terminal nodes results in the lowest cross-validation error rate, with 133 cross-validation errors. However, the original tree

without pruning already has 8 nodes, so the pruned tree is the same as the original tree. We also calculate the confusion matrix of the six-node tree, and get the same classification result with the eight-node tree, which means that the pruning doesn't improve the accuracy on this dataset.

Section 5. Conclusion and Future Works

In this experiment, we implemented different algorithms. According to our result in table in Section 3, we get very good performance in some algorithms, that is SVM with RBF and polynomial kernels, AdaBoost and XGBoost along with good computation duration and accuracy.

As for the pros and cons of SVM, we can find that if there is a clear margin of separation between classes (like our dataset), then SVM will work very well. However, SVM is not suitable for an even larger dataset and it does not perform well when a dataset has a large amount of noise (i.e. target classes are overlapping). Thus, it might not be a good choice in cases where there are more predictors and the number of predictors of the test dataset exceeds the training dataset.

AdaBoost is robust for overfitting low-noise data sets, and it has only a few hyperparameters that need to be tuned to improve model performance, in contrast to XGBoost. In addition, AdaBoost is easy to understand and visualize but relatively time consuming.

For further results, all of the algorithms examined in this experiment have nearly the same performances, in other words, there is no very outstanding algorithm among the ones we have tested. Therefore, a proposal of building up an ensemble model to make different algorithms work with each other.

Just like the ideology of AdaBoost algorithm: combine many weak classifiers into a stronger one. In the ensemble method, maybe we can boost a series of strong classifiers into an extremely stronger one. However, this is just an ideology. The combination of multiple classifiers will not always perform better than one individual classifier.

The results indicate that more features can be added to the dataset in future work, which could improve the performance of these models. Maybe we can further extrapolate that to combine machine learning methods with some other anti-phishing approaches to fortify the defense line of the Internet.

Reference

[1] Doyen Sahoo. Malicious URL Detection using Machine Learning: A survey. Vol. 1, No. 1. Aug 2019.

[2] Neda Abdelhamid. Phishing Detection Based Associative Classification Data Mining. Mar 2014.

[3] Vahid Shahrivari, Mohammad M. Darabi. Phishing Detection Using Machine Learning Techniques. Sep 2020.