DS Journal of Cyber Security
Volume 2 Issue 3, 15-25, Jul-Sep 2024
ISSN: 2584-0665/ https://doi.org/10.59232/CYS-V2I3P102

Original Article

Phishing Attack Detection Using Machine Learning

Olaniyi. A. Ayeni[1], Hope O. Akinyemi[2]

[1,2]*Department of Cybersecurity, Federal University of Technology, Akure, Nigeria.*

oaayeni@futa.edu.ng

Received: 15 July 2024; Revised: 20 August 2024; Accepted: 18 September 2024; Published: 30 September 2024;

Abstract - Phishing acquires sensitive information, like login credentials, i.e. usernames, passwords or other security tokens, card information, etc, from users. In most cases, Phishing does not require high technicality, making it the third most performed attack according to multiple sources. Despite many solutions being proposed to completely eradicate or at least mitigate about 80% of Phishing attacks, Phishing continues to be prevalent. Vishakha P.R. & Sahil S.J. (2020) were unable to detect web pages automatically, and the systems were limited to system applications alone. Therefore, this work aims to design a system for detecting phishing attacks, Implement the design, and evaluate the algorithm's performance. Jupyter Notebook was the medium used for analysis. The two algorithms used were two-class logistic detection and Naïve Bayes. The dataset (phishing.csv) obtained from Kaggle was divided into two datasets to train and test in the ratio 80:20. The result of two class logistic regression was 94.2% compared to 85% test accuracy by the Naïve Bayes algorithm.

Keywords - Phishing detection, Naïve Bayes, Machine Learning, Logistic regression, Cyber production.

1. Introduction

Phishing is an internet attack mostly propagated by email. It has grown rampant in recent years, intending to deceive the victim into accepting that the email is legit and urgently requires attention like a request from their bank, from their place of work, or An urgent question from their boss to download an attachment. It can be very difficult to detect, especially if the targeted victims are non-technical staff members.

Phishing has thrived due to the urgency of its message; for example, a newly employed staff member receives a mail from his direct boss asking him to re-upload a document he sent earlier. Being eager to impress his new boss, the new staff might not necessarily think of confirming over a voice call before getting himself phished. In other cases, the big figures in an organization, such as the executive officers and directors, can be the target since they would be of more financial advantage; this is a type of phishing called whaling.

2. Review of Related Works

After thoroughly reviewing multiple research papers, the contributions made by various researchers are discussed. Vishakha P.R. & Sahil S.J. (2020) [24] created a framework to detect phishing sites, and this framework makes use of efficient techniques to detect phishing sites. Certain important characteristics, such as the URL and Domain Identity, can be used to identify phishing sites. The finest security mechanism feasible is what the suggested system aims to deliver, giving those who use online transactions the trust they need. The goal of this system is to create a generally regarded system that plays a crucial part in security issues for the banking era.

Nalin Asanka, Gamagedara Arachchilage, and SteveLove (2013) [5] created a paradigm for game creation based on the Technology Threat Avoidance Theory to counteract phishing attempts. To prevent phishing attacks, the framework made use of enhanced avoidance behaviours by users. The goal of this project was to create a framework for game design that would motivate users to avoid phishing sites and improve their avoidance behaviour. This methodology for game design was employed not only to stop phishing scams but also to stop other malevolent behaviours and attacks.

Mohith GH et al. (2020) [3] Due to the failure of other methods, Mohith G. et al. introduced methods into the development of browsers named 'Embedded Phishing Detection Browser' (EPDB), this detection method to maintain the interaction of users alongside ensuring Optimal security of an individual system or network as the case may be. The novelty of the EPDB (Embedded Phishing Detection Browser) approach was the new browser architecture, developed by manipulating old architecture, developing a new structure named "Intelligent Engine". This allows phishing detection on websites easily and in real time.

Colin Whittaker et al. (2009): [2] This paper was aimed at developing a phishing detection medium by describing the design and performance characteristics of a scalable machine learning classifier. This framework was basically developed to automatically detect phishing websites and get recognized and blacklisted with Google, and it is maintained and sustained automatically.

Yan Chen et al. (2021): [4] The significant aspects (also known as trust calibrators) in calibrating trust in phishing-website detection systems and the ensuing results are identified theoretically in this paper. The trust calibration model (abbreviated TC model for short) for phishing website detection systems was developed using the Automation Trust and Reliance (ATR) theoretical framework as a kernel theory.

Niels Provos et al. (2010) [6]: Since earlier detection strategies were shown to be insufficiently reliable, this research focuses on enhancing URL-based detection of phishing sites. Although strategies that depend on obtaining content that a URL points to may yield higher detection rates, it is crucial to remember that these strategies may have unintended consequences.

N.Swapna Goud et al. (2021)[8]: The goal of this study was to identify the key characteristics that determine whether or not a link is for phishing. Here, ensemble algorithms are utilized to tackle the feature selection problem for phishing website detection. Ensemble algorithms are well-liked for their reliable outcomes. The steps undertaken in feature engineering aid in the model's construction of the explanatory qualities, which are essential for model training. Because authors have not widely used feature engineering, this research has a great deal of academic relevance.

Abedin, N. F. et al. (2020): This research study utilized three machine learning algorithms. In order to prevent Zero-Day attacks, these models are trained using URL-based features utilizing recommended software that differentiates between phishing and legitimate websites by looking at the URL. Based on the data, the random forest classifier achieved 97% F1 Score, 99% recall, and 97% precision. Unlike earlier research, the proposed model employs only the URL for analysis and does not rely on any other sources, making it fast and efficient.

Hossain, S. et al. (2020): The goal of this work is to determine the most workable means for recognizing one of the prevalent internet attacks. This would enable quicker identification and removal of such websites, making web browsing safer and more secure for everyone. In order to do this, we provide a detailed description of each methodology we examine and employ several evaluation methods to create a visual representation of their performance. After comparing and contrasting all of these methods, we have concluded in this study that the Random Forest Classifier is the most effective detection in phishing detection.

Salahdine, F., et al (2021). In this study, a machine learning-based technique for phishing attack detection was presented. We collected and investigated more than 4000 phishing emails that were directed towards the University of North Dakota's email system. We were able to model these attacks by selecting ten relevant variables and assembling a substantial dataset. This dataset served as training, validation, and testing data for the machine learning techniques. The following four metrics have been used to assess performance: accuracy, miss-detection probability, false alarm probability, and detection probability. The outcomes of the experiment demonstrate that an artificial neural network can be used to get improved detection.

Table 1. Differet researches and used techniques

Author	Method used
Vishakha P.R. & Sahil S.J. (2020)	URL and Doman Identirty
Hossain, S., et al (2020)	Random Forest
Salahdine, F., et al (2021	Rule-Based, White and Blacklist, Heuristic, and Hybrid
Abedin, N. F., et al (2020)	Random Forest
N.Swapna Goud et al (2021)	Feature Engineering
Mohith Gowda HR et al. (2020)	Embedded Phishing Detector

3. Motivation/ Research Gap

The limitation of previous works by Vishakha P.R. & Sahil S.J. (2020) was the inability of the findings to automatically detect the web page and the system being limited to system applications alone. - The rate of phishing in recent years has escalated more than its prediction. Its ease and non-technicality have made it an easier option for internet fraudsters. The increase in Internet activities over the last decade has also encouraged more cyberattacks, with most Institutions and organizations migrating their information to the cloud. Internet fraudsters have an easier and broader surface to exploit. This research, therefore, aimed to:

1. Design a system for the detection of phishing attack.
2. Implement the design in (a).
3. Evaluate the developed system.

3.1. Machine Learning

Machine learning seeks to copy human reasoning and provides practical approaches to problems that would otherwise be impossible to solve by hand. It tackles the problem of creating computers that automatically get better with use. It is one of the technical areas that is expanding the fastest right now. Machine learning is currently a go-to technique in artificial intelligence in the creation of software for applicable tasks.

3.2. Naïve Bayes

Based on the assumption of feature independence, naïve Bayes is a type of probability-oriented technique from Bayes' theorem. To make predictions using the predict function, the model is trained using a training dataset.

3.3. Logistic Regression

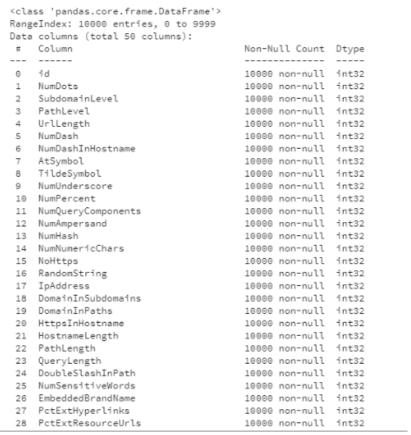
A technique created to solve classification issues. A categorical target variable presents a classification learning challenge. To predict the likelihood that a new example will belong to one of the target classes, logistic regression aims to map a function from the dataset's features to the targets and puts together a training set of observations with $p=(n_t,p_t)$, k=1,…,k. A computer program called a learning algorithm receives the values from the user x_i and uses them to generate outputs in response to the inputs. The ability of the learning algorithm to adjust its input/output relationship based on variations between the generated and original outputs is one of its features. We call this approach "learning by example." The fake and real outputs will be almost sufficient for all inputs that are likely used in practice as the process of learning is finished.

3.4. Decision Tree

One formalism for representing maps is this one. It creates an inverted tree with a root node, internal nodes, and leaf nodes by dividing a population into sections that resemble branches. It can handle big, complex datasets because it is non-parametric and does not require a convoluted parametric framework. The datasets can be separated into training and validation datasets once the size of the study sample is sufficiently large. A decision tree can be constructed from a set of instances using a divide-and-conquer strategy. If each instance belongs to the same class, then the tree is a leaf with the class indicated. If not, a test with different results for at least two of the instances is partitioned using this result.

3.5. Dataset

The datasets used for this project were acquired from the Mendeley Dataset, which is a collection of publicly available websites which the features presented in the dataset were compiled from. It is compiled in a .csv file format. The datasets comprise the features from the collection of the website and consist of about 1000 rows and 50 columns. Its datatypes comprise 3 floats and 47 integers. It occupies a memory space of about 1.4MB. It consists of legitimate and phishing sites and includes features such as URL length, iFrame and others. These data consist of various instances of legitimate and phishing sites. Each is denoted by characteristics that depict whether the website is phished or otherwise. The feature 'CLASS_LABEL' represents the phishing label using 1 and the Legitimate label, 0. The data is an input for the learning process.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       10000 non-null  int32
 1   NumDots                  10000 non-null  int32
 2   SubdomainLevel           10000 non-null  int32
 3   PathLevel                10000 non-null  int32
 4   UrlLength                10000 non-null  int32
 5   NumDash                  10000 non-null  int32
 6   NumDashInHostname        10000 non-null  int32
 7   AtSymbol                 10000 non-null  int32
 8   TildeSymbol              10000 non-null  int32
 9   NumUnderscore            10000 non-null  int32
 10  NumPercent               10000 non-null  int32
 11  NumQueryComponents       10000 non-null  int32
 12  NumAmpersand             10000 non-null  int32
 13  NumHash                  10000 non-null  int32
 14  NumNumericChars          10000 non-null  int32
 15  NoHttps                  10000 non-null  int32
 16  RandomString             10000 non-null  int32
 17  IpAddress                10000 non-null  int32
 18  DomainInSubdomains       10000 non-null  int32
 19  DomainInPaths            10000 non-null  int32
 20  HttpsInHostname          10000 non-null  int32
 21  HostnameLength           10000 non-null  int32
 22  PathLength               10000 non-null  int32
 23  QueryLength              10000 non-null  int32
 24  DoubleSlashInPath        10000 non-null  int32
 25  NumSensitiveWords        10000 non-null  int32
 26  EmbeddedBrandName        10000 non-null  int32
 27  PctExtHyperlinks         10000 non-null  int32
 28  PctExtResourceUrls       10000 non-null  int32
```

Fig. 1 Visual representation of the dataset used

4. Results

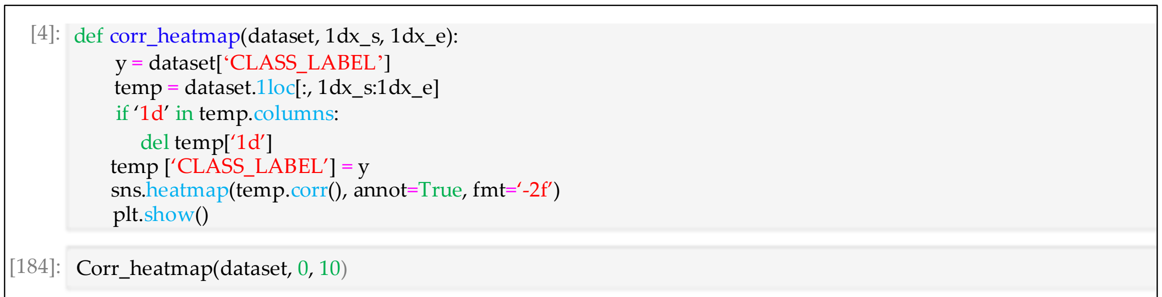
4.1. Data Preprocessing

In order to appropriately set up the data as input for a particular Data Mining algorithm, a series of techniques known as data pretreatment is applied. Preprocessing data is typically a required step. It creates fresh data that matches a Data Mining method from previously meaningless data. First of all, poorly prepared data may not be received for operation by the Data Mining method, or it will undoubtedly disclose mistakes while it is running. In the best of circumstances, the algorithm will provide findings, but they will not make sense or be regarded as reliable information. The data is transformed or combined in this preparation stage so that the mining process outcome can be used or could be more effective. A few of the subtasks that fall under the category of data transformation are feature building, normalization, discretization, generalization, aggregation, and smoothing.

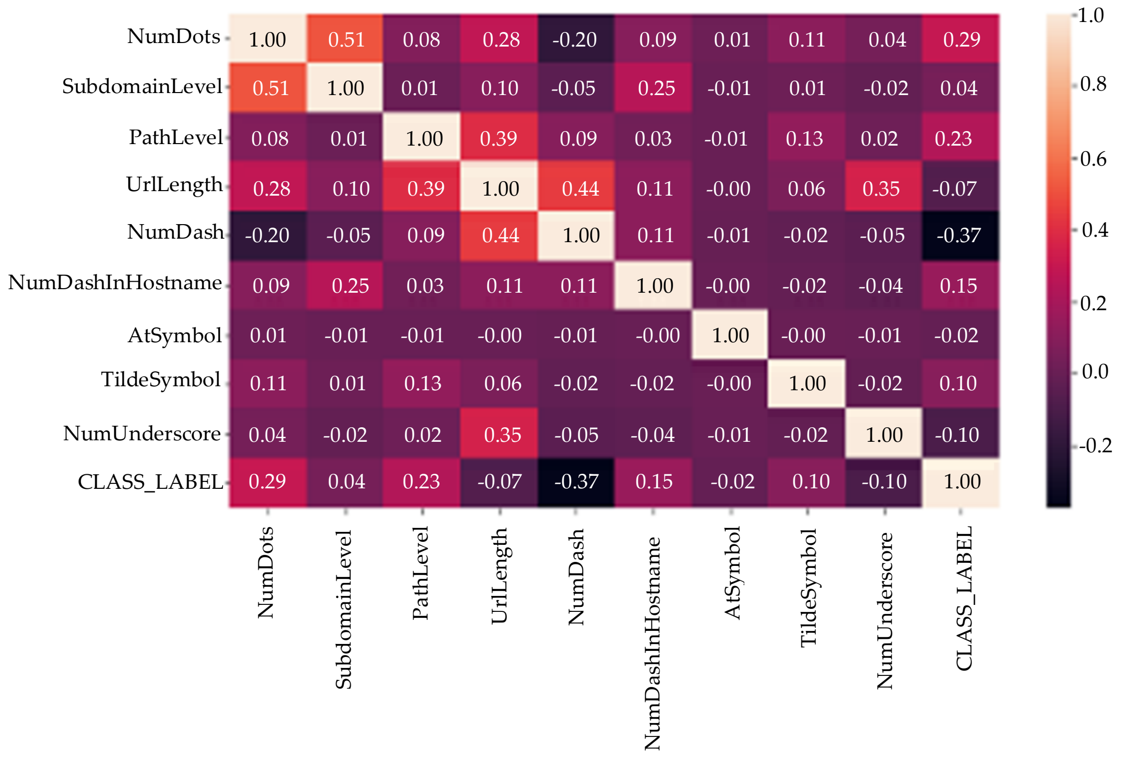
Most of the procedures in data transformation, including data cleaning, will be broken out into separate tasks since they are referred to as a broad data preparation family.

4.2. Training and Result

Here, we used a data-oriented means to train and learn, which essentially takes several N features for model training the model and the metrics are re-evaluated. This uses a logistic regression model to perform a repetitive training process with the goal of determining the perfect amount of features that can be utilized to find the model that best fits without many adjustments.



```
[4]:  def corr_heatmap(dataset, 1dx_s, 1dx_e):
          y = dataset['CLASS_LABEL']
          temp = dataset.1loc[:, 1dx_s:1dx_e]
          if '1d' in temp.columns:
              del temp['1d']
          temp ['CLASS_LABEL'] = y
          sns.heatmap(temp.corr(), annot=True, fmt='-2f')
          plt.show()

[184]: Corr_heatmap(dataset, 0, 10)
```



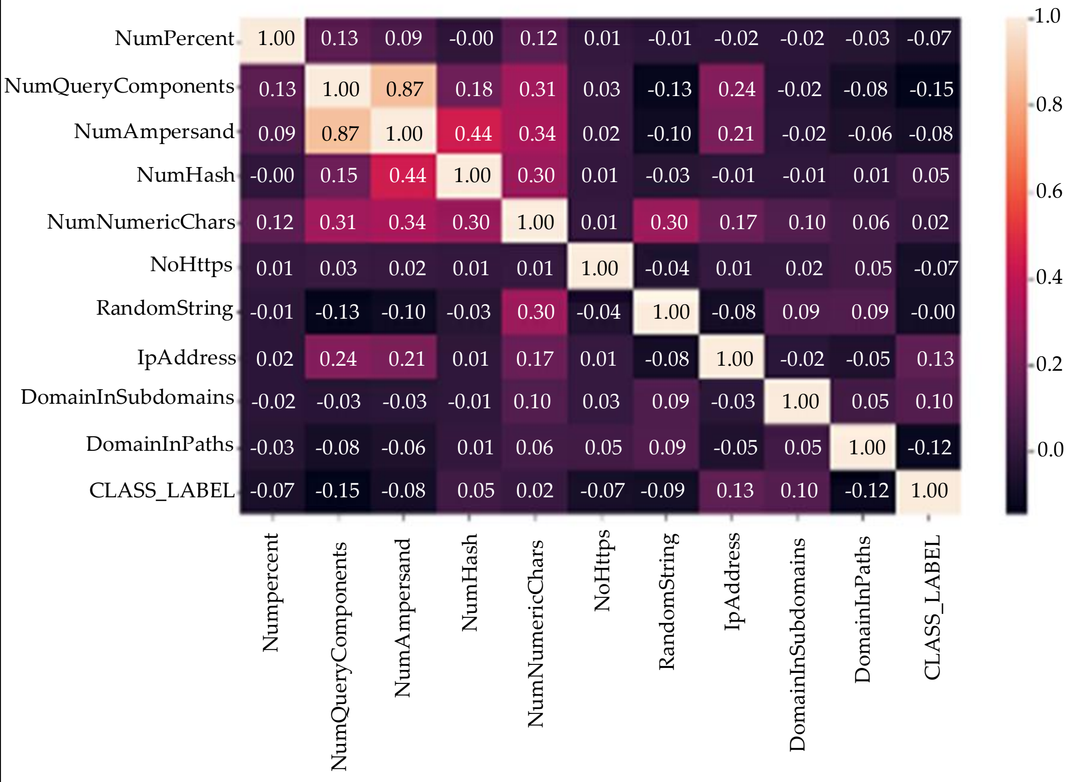
	NumDots	SubdomainLevel	PathLevel	UrlLength	NumDash	NumDashInHostname	AtSymbol	TildeSymbol	NumUnderscore	CLASS_LABEL
NumDots	1.00	0.51	0.08	0.28	-0.20	0.09	0.01	0.11	0.04	0.29
SubdomainLevel	0.51	1.00	0.01	0.10	-0.05	0.25	-0.01	0.01	-0.02	0.04
PathLevel	0.08	0.01	1.00	0.39	0.09	0.03	-0.01	0.13	0.02	0.23
UrlLength	0.28	0.10	0.39	1.00	0.44	0.11	-0.00	0.06	0.35	-0.07
NumDash	-0.20	-0.05	0.09	0.44	1.00	0.11	-0.01	-0.02	-0.05	-0.37
NumDashInHostname	0.09	0.25	0.03	0.11	0.11	1.00	-0.00	-0.02	-0.04	0.15
AtSymbol	0.01	-0.01	-0.01	-0.00	-0.01	-0.00	1.00	-0.00	-0.01	-0.02
TildeSymbol	0.11	0.01	0.13	0.06	-0.02	-0.02	-0.00	1.00	-0.02	0.10
NumUnderscore	0.04	-0.02	0.02	0.35	-0.05	-0.04	-0.01	-0.02	1.00	-0.10
CLASS_LABEL	0.29	0.04	0.23	-0.07	-0.37	0.15	-0.02	0.10	-0.10	1.00

Fig. 2 Correlation for the first 10 columns

As shown in the legend above, 0.5 is the average correlation in the positive direction; for the first 10 columns, the highest value is NumDots at 0.29, which is far from an average correlation with the label.

There are still no strong or even medium-level correlation features, both positive and negative, in the next 10 rows with CLASS_LABEL. The highest being 0.13 from ip address.

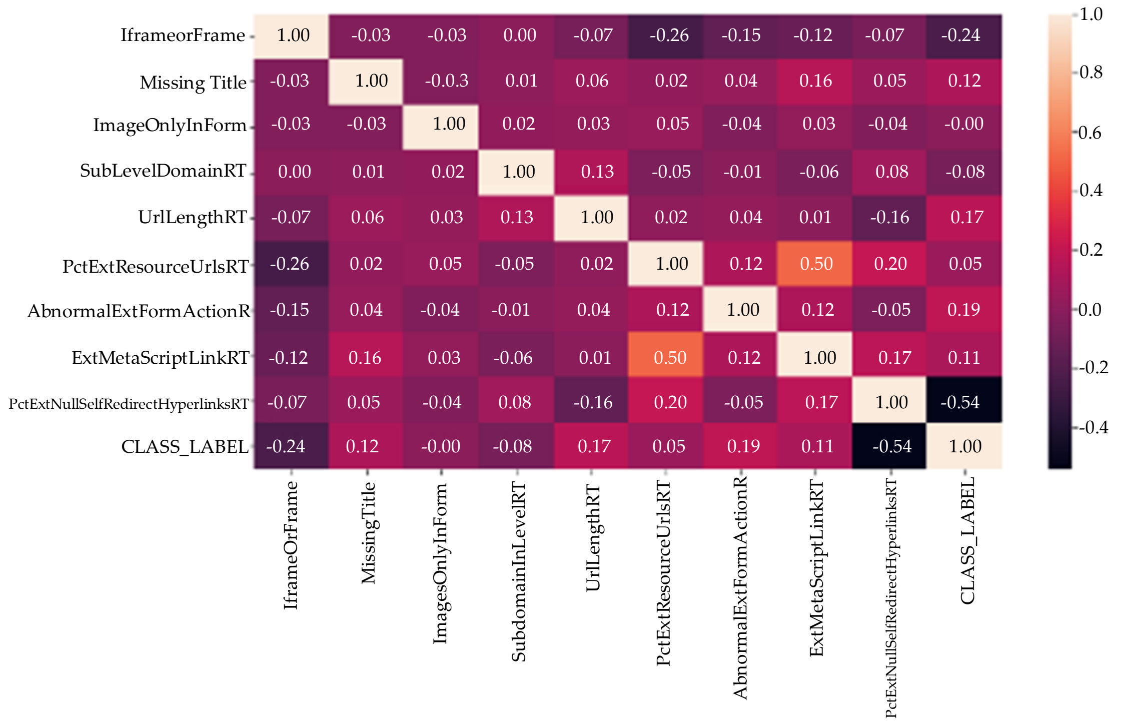
```
[185]: Corr_heatmap(dataset, 10, 20)
```



	Numpercent	NumQueryComponents	NumAmpersand	NumHash	NumNumericChars	NoHttps	RandomString	IpAddress	DomainInSubdomains	DomainInPaths	CLASS_LABEL
NumPercent	1.00	0.13	0.09	-0.00	0.12	0.01	-0.01	-0.02	-0.02	-0.03	-0.07
NumQueryComponents	0.13	1.00	0.87	0.18	0.31	0.03	-0.13	0.24	-0.02	-0.08	-0.15
NumAmpersand	0.09	0.87	1.00	0.44	0.34	0.02	-0.10	0.21	-0.02	-0.06	-0.08
NumHash	-0.00	0.15	0.44	1.00	0.30	0.01	-0.03	-0.01	-0.01	0.01	0.05
NumNumericChars	0.12	0.31	0.34	0.30	1.00	0.01	0.30	0.17	0.10	0.06	0.02
NoHttps	0.01	0.03	0.02	0.01	0.01	1.00	-0.04	0.01	0.02	0.05	-0.07
RandomString	-0.01	-0.13	-0.10	-0.03	0.30	-0.04	1.00	-0.08	0.09	0.09	-0.00
IpAddress	0.02	0.24	0.21	0.01	0.17	0.01	-0.08	1.00	-0.02	-0.05	0.13
DomainInSubdomains	-0.02	-0.03	-0.03	-0.01	0.10	0.03	0.09	-0.03	1.00	0.05	0.10
DomainInPaths	-0.03	-0.08	-0.06	0.01	0.06	0.05	0.09	-0.05	0.05	1.00	-0.12
CLASS_LABEL	-0.07	-0.15	-0.08	0.05	0.02	-0.07	-0.09	0.13	0.10	-0.12	1.00

Fig. 3 Correlation between columns 10 and 20

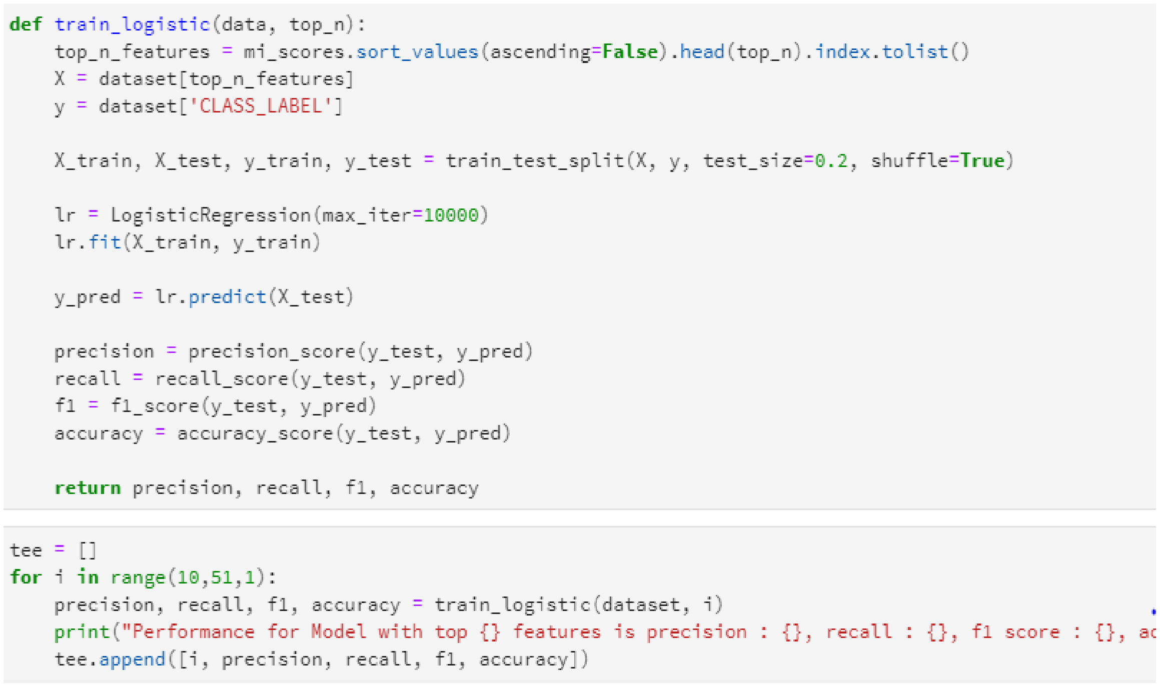
```
Corr_heatmap(dataset, 40, 50)
```



	IframeOrFrame	MissingTitle	ImagesOnlyInForm	SubdomainInLevelRT	UrlLengthRT	PctExtResourceUrlsRT	AbnormalExtFormActionR	ExtMetaScriptLinkRT	PctExtNullSelfRedirectHyperlinksRT	CLASS_LABEL
IframeorFrame	1.00	-0.03	-0.03	0.00	-0.07	-0.26	-0.15	-0.12	-0.07	-0.24
Missing Title	-0.03	1.00	-0.0.3	0.01	0.06	0.02	0.04	0.16	0.05	0.12
ImageOnlyInForm	-0.03	-0.03	1.00	0.02	0.03	0.05	-0.04	0.03	-0.04	-0.00
SubLevelDomainRT	0.00	0.01	0.02	1.00	0.13	-0.05	-0.01	-0.06	0.08	-0.08
UrlLengthRT	-0.07	0.06	0.03	0.13	1.00	0.02	0.04	0.01	-0.16	0.17
PctExtResourceUrlsRT	-0.26	0.02	0.05	-0.05	0.02	1.00	0.12	0.50	0.20	0.05
AbnormalExtFormActionR	-0.15	0.04	-0.04	-0.01	0.04	0.12	1.00	0.12	-0.05	0.19
ExtMetaScriptLinkRT	-0.12	0.16	0.03	-0.06	0.01	0.50	0.12	1.00	0.17	0.11
PctExtNullSelfRedirectHyperlinksRT	-0.07	0.05	-0.04	0.08	-0.16	0.20	-0.05	0.17	1.00	-0.54
CLASS_LABEL	-0.24	0.12	-0.00	-0.08	0.17	0.05	0.19	0.11	-0.54	1.00

Fig. 4 Correlation between columns 40 and 50

The legend above shows results for rows 40 – 50, At -0.54, PctExtNullSelfRedirectHyperlinksRT is the sole column within this group that exhibits some association with labels, albeit negatively this time. This could indicate that as the percentage of null self-redirect hyperlinks increases, so does the likelihood of phishing attacks. Training the logistic model would help easily predict if an email is phishing or spam by rightly considering historical data and, in this case, our dataset. We started the loop from 10 as we will train from the 10th feature to the 50th feature to find the optimal number of features needed for this problem.

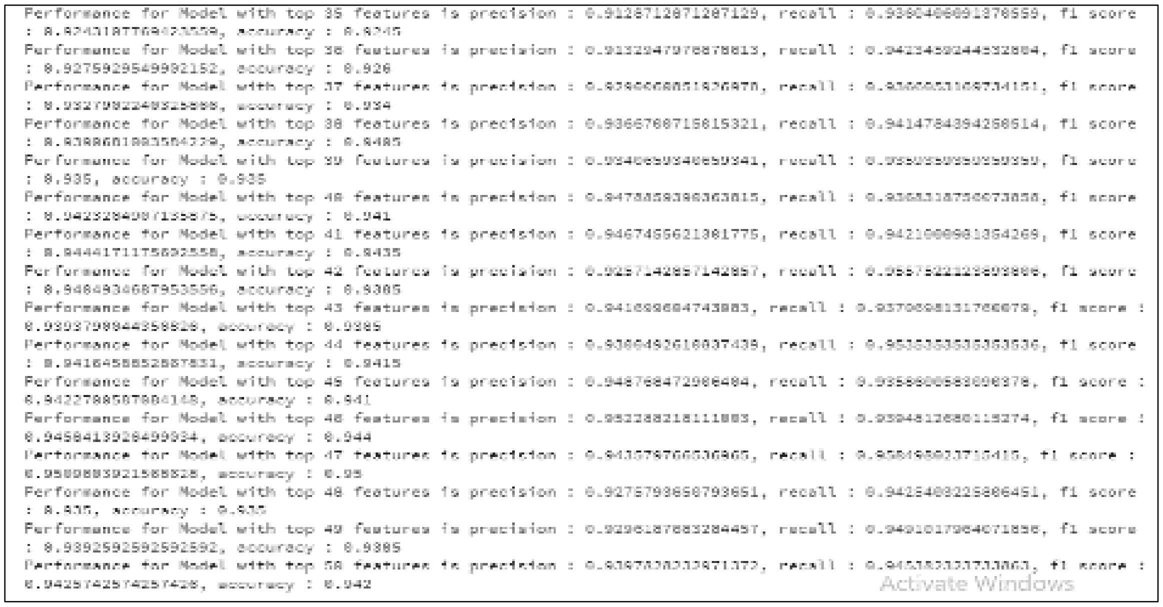


```python
def train_logistic(data, top_n):
    top_n_features = mi_scores.sort_values(ascending=False).head(top_n).index.tolist()
    X = dataset[top_n_features]
    y = dataset['CLASS_LABEL']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

    lr = LogisticRegression(max_iter=10000)
    lr.fit(X_train, y_train)

    y_pred = lr.predict(X_test)

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    return precision, recall, f1, accuracy
```

```python
tee = []
for i in range(10,51,1):
    precision, recall, f1, accuracy = train_logistic(dataset, i)
    print("Performance for Model with top {} features is precision : {}, recall : {}, f1 score : {}, ac
    tee.append([i, precision, recall, f1, accuracy])
```

Fig. 5 Training of the dataset

The training produced a considerably high level of accuracy and precision, showing that its phishing detection ability would almost accurately detect when being phished.



```
Performance for Model with top 35 features is precision : 0.9128712871287129, recall : 0.9380406091370559, f1 score
: 0.9243187769472559, accuracy : 0.9245
Performance for Model with top 36 features is precision : 0.9133247976878813, recall : 0.9423459244532804, f1 score
: 0.9275929549902152, accuracy : 0.926
Performance for Model with top 37 features is precision : 0.9296660851826970, recall : 0.9366651169734151, f1 score
: 0.9327902240325866, accuracy : 0.934
Performance for Model with top 38 features is precision : 0.9366768715815321, recall : 0.9414784394250514, f1 score
: 0.9390681003584229, accuracy : 0.9485
Performance for Model with top 39 features is precision : 0.9340659340659341, recall : 0.9350359350359350, f1 score
: 0.935, accuracy : 0.935
Performance for Model with top 40 features is precision : 0.9478458390363815, recall : 0.9368318756073858, f1 score
: 0.9423284907135875, accuracy : 0.941
Performance for Model with top 41 features is precision : 0.9467455621301775, recall : 0.9421000981354269, f1 score
: 0.9444171175692558, accuracy : 0.9435
Performance for Model with top 42 features is precision : 0.9257142857142857, recall : 0.9557522123893806, f1 score
: 0.9404934687953556, accuracy : 0.9385
Performance for Model with top 43 features is precision : 0.9416996047430830, recall : 0.9370698131760079, f1 score :
0.9393799844358820, accuracy : 0.9385
Performance for Model with top 44 features is precision : 0.9306492618037438, recall : 0.9535353535353536, f1 score
: 0.9416458852867831, accuracy : 0.9415
Performance for Model with top 45 features is precision : 0.9487684729064040, recall : 0.9356800683090378, f1 score :
0.9422788587084148, accuracy : 0.941
Performance for Model with top 46 features is precision : 0.9522388218111883, recall : 0.9394813608115274, f1 score :
0.9458413926499034, accuracy : 0.944
Performance for Model with top 47 features is precision : 0.9435787666530965, recall : 0.9584980237154150, f1 score :
0.9509803921568628, accuracy : 0.95
Performance for Model with top 48 features is precision : 0.9275793650793651, recall : 0.9425403225806451, f1 score
: 0.935, accuracy : 0.935
Performance for Model with top 49 features is precision : 0.9296187683284457, recall : 0.9491017964071856, f1 score
: 0.9392592592592592, accuracy : 0.9385
Performance for Model with top 50 features is precision : 0.9397828232971372, recall : 0.9451823237133863, f1 score :
0.9425742574257426, accuracy : 0.942
```

Fig. 6 Training result

The model was unstable as more features were added during training, so we found the 39th feature to be the one with the best metrics and all-around performance; it came closest with the lowest score of 0.947162. The result is represented graphically below:

```
sns.lineplot(x='num_of_features', y='precision', data=df, label='Precision Score')
sns.lineplot(x='num_of_features', y='recall', data=df, label='Recall Score')
sns.lineplot(x='num_of_features', y='fl_score', data=df, label='F1 Score')
sns.lineplot(x='num_of_features', y='accuracy', data=df, label='Acc Score')
```

```
<AxesSubplot:xlabel='num_of_features', ylabel='precision'>
```

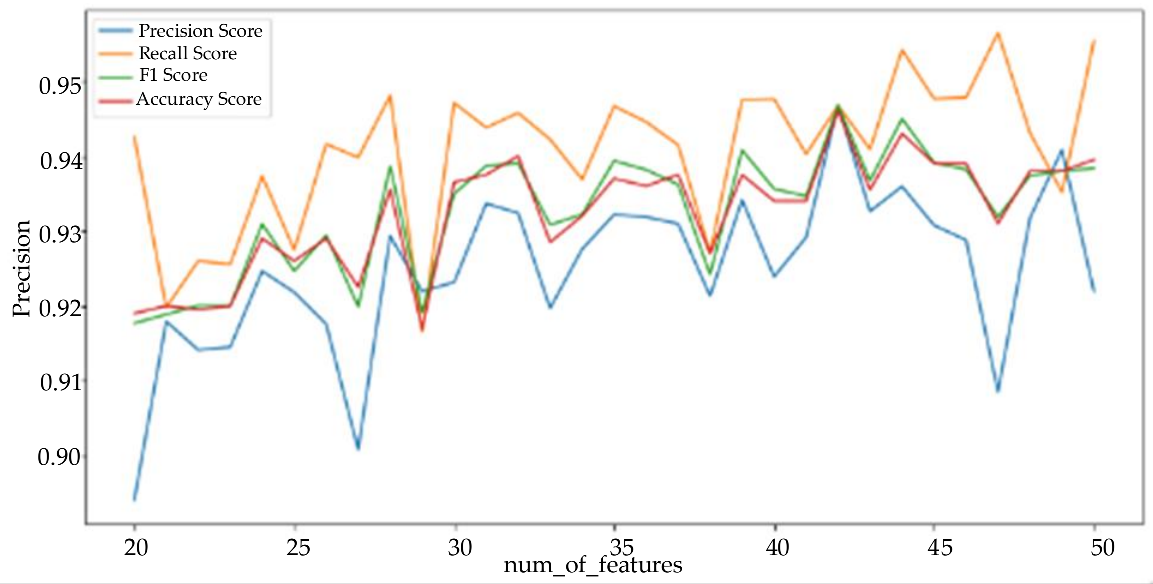


Fig. 7 Graphical representation of data result

Additionally, recall is also constantly performing well, but our model tends to have problems with precision scores. Compared with the Naïve Bayes model,

```
naive_bayes = GaussianNB()
naive_bayes = naive_bayes.fit(X_train, y_train)
pred = naive_bayes.predict(X_train)

print('confusion matrix plot for training data:')
cm = confusion_matrix(y_train, pred)
display_cm = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=naive_bayes.classes_)
display_cm.plot()
plt.show()

print(f'\n\ntraining accuracy of model: {round(accuracy_score(y_train, pred)*100, 4)}%')
```

```
confusion matrix plot for training data:
```

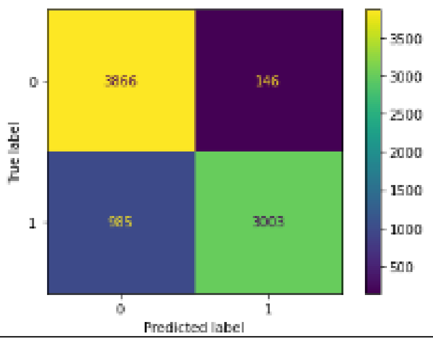
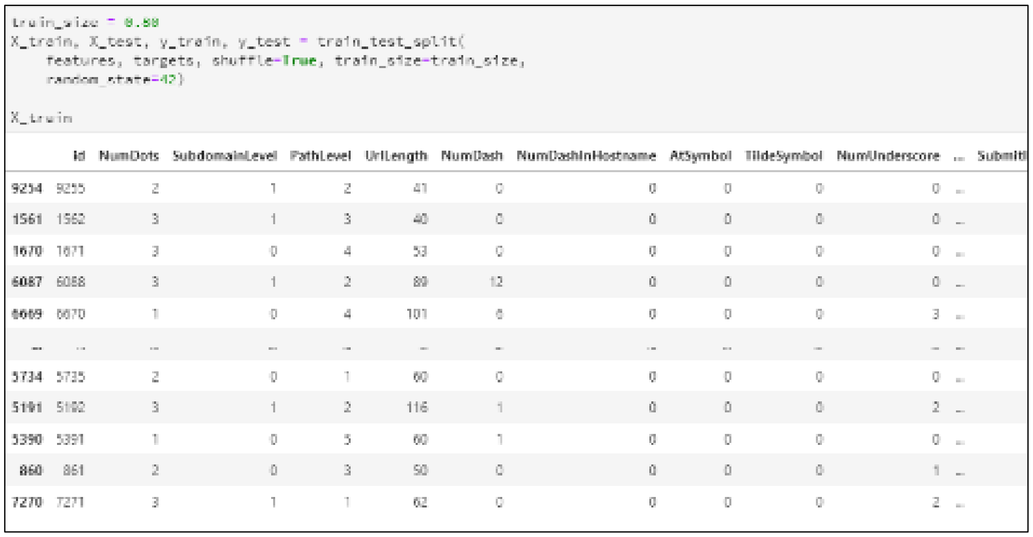


Fig. 8 Comparative modelling

Since Machine Learning models work on past observations, the accuracy of the model could be subject to its exposure to training data. Meanwhile, the testing dataset tests the performance of the trained data to determine if it needs more training or has been trained effectively enough, so using the 20:80 proportion is not new.



```python
train_size = 0.80
X_train, X_test, y_train, y_test = train_test_split(
    features, targets, shuffle=True, train_size=train_size,
    random_state=42)

X_train
```

```
        Id  NumDots  SubdomainLevel  PathLevel  UrlLength  NumDash  NumDashInHostname  AtSymbol  TildeSymbol  NumUnderscore  ...  Submitt
9254  9255        2               1          2         41        0                  0         0            0              0  ...
1561  1562        3               1          3         40        0                  0         0            0              0  ...
1670  1671        3               0          4         53        0                  0         0            0              0  ...
6087  6088        3               1          2         89       12                  0         0            0              0  ...
6669  6670        1               0          4        101        6                  0         0            0              3  ...
 ...   ...      ...             ...        ...        ...      ...                ...       ...          ...            ...  ...
5734  5735        2               0          1         60        0                  0         0            0              0  ...
5191  5192        3               1          2        116        1                  0         0            0              2  ...
5390  5391        1               0          5         60        1                  0         0            0              0  ...
 860   861        2               0          3         50        0                  0         0            0              1  ...
7270  7271        3               1          1         62        0                  0         0            0              2  ...
```

Fig. 9 Bayes training

5. Evaluation of Previous Work

Comparing the machine learning performance with similar authors.

Table 2. Comparison of previous works

Author	Recall	Precision	F1-score	Accuracy
This work, 2023	94.5%	93.9%	94.2%	94.2%
Basnet, R., Mukkamala, S., & Sung, A. H. (2014)	97%	96.6%	96.8%	96.8%
Niels P., Dean M., Panayiotis M., et al. (2010)	94%	96%	95%	95%

6. Conclusion

In this paper, We suggested utilizing machine learning to detect phishing attacks. Using the dataset, three classifiers are trained and evaluated. A parametric research is carried out for each classifier, and the best findings are reported for assessment. Among the evaluated algorithms, Naïve Bayes demonstrated the best accuracy at 94.2%. Comparatively, N. F. Abedin et al. (2020) in "Phishing Attack Detection using Machine Learning Classification Techniques", the performance of the three widely used machine learning classifiers is compared. Among these three classifiers, random forest performance is the highest, with a precision of 97%.

In the future, features will be changed to increase accuracy. Systems based on deep learning models and neural networks that can handle vast amounts of data can be built to identify phishing attacks from logged datasets. Future research to increase the system's accuracy will also take feature reduction approaches into consideration.

References

[1] Sadia Afroz, and Rachel Greenstadt, "Phishzoo: Detecting Phishing Websites by Looking at Them," *2011 IEEE Fifth International Conference on Semantic Computing*, Palo Alto, USA, pp. 368-375, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[2] Olaniyi Abiodun Ayeni, "A Supervised Machine Learning Algorithm for Detecting Malware," *Journal of Internet Technology and Secured Transactions*, vol. 10, no.1, pp. 764-769, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[3] Ram Basnet, Srinivas Mukkamala, and Andrew H. Sung, "Detection of Phishing Attacks: A Machine Learning Approach," *Soft Computing Applications in Industry*, vol. 226, pp. 373-383, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[4] Ekaba Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, 1st ed., Apress Berkeley, CA, pp. 243-250, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Yan Chen, "Trust Calibration of Automated Security IT Artifacts: A Multi-Domain Study of Phishing-Website Detection Tools," *Information & Management*, vol. 58, no. 1, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Juan Chen, and Chuanxiong Guo, "Online Detection and Prevention of Phishing Attacks," *2006 First International Conference on Communications and Networking in China*, Beijing, China, pp. 1-7, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[7] Sujata Garera, "A Framework for Detection and Measurement of Phishing Attacks," *WORM '07: Proceedings of the 2007 ACM workshop on Recurring Malcode*, pp. 1-8, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[8] N. Swapna Goud, and Anjali Mathur, "Feature Engineering Framework to Detect Phishing Websites Using URL Analysis," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 7, pp. 295-303, 2021. [Google Scholar]

[9] Trevor Hastie, Jerome Friedman, and Robert Tibshirani, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," 1st ed., Springer New York, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[10] Mohith Gowda H.R. et al., "Development of Anti-Phishing Browser Based on Random Forest and Rule of Extraction Framework," *Cybersecurity*, vol. 3, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[11] Mazharul Islam, and Nihad Karim Chowdhury, "Phishing Websites Detection Using Machine Learning-Based Classification Techniques," *International Conference on Advanced Information and Communication Technology*, Chittagong, Bangladesh, vol. 10, no. 9, pp. 4393-4402, 2016. [Google Scholar]

[12] Ankit Kumar Jain, and B.B. Gupta, "Towards Detection of Phishing Websites on Client-Side Using Machine Learning-Based Approach," *Telecommunication Systems*, vol. 68, pp. 687-700, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[13] Markus Jakobsson, "Modeling and Preventing Phishing Attacks," *Financial Cryptography and Data Security*, vol. 5, pp. 1-19, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[14] M.I. Jordan, and T.M. Mitchell, "Machine Learning: Trends, Perspectives, and Prospects," *Science*, vol. 349, no. 6245, pp. 255-260, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[15] R. Kiruthiga, and D. Akila, "Phishing Websites Detection Using Machine Learning," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2S11, pp. 111-114, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[16] Arun D. Kulkarni, and Leonard L. III Brown, "Phishing Websites Detection Using Machine Learning," *Computer Science Faculty Publications and Presentations*, pp. 8-13, 2019. [Google Scholar] [Publisher Link]

[17] Niklas Lavesson, and Paul Davidsson, "Evaluating Learning Algorithms and Classifiers," *International Journal of Intelligent Information and Database Systems*, vol. 1, no. 1, pp. 37-52, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[18] Scott Menard, *Applied Logistic Regression Analysis*, 2nd ed., Sage Publications, New Delhi, India, 2022. [Google Scholar]

[19] H.R. Mohith Gowda et al., "Development of Anti-Phishing Browser Based on Random Forest and Rule of Extraction Framework," *Cybersecurity*, vol. 3, pp. 1-14, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] J.R. Quinlan, "Learning Decision Tree Classifiers," *ACM Computing Surveys*, vol. 28, no. 1, pp. 71-72, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[21] Jiangtao Ren et al., "Naive Bayes Classification of Uncertain Data," *2009 Ninth IEEE International Conference on Data Mining*, Miami Beach, FL, USA, pp. 944-949, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[22] S.R. Safavian, and D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660-674, 1991. [CrossRef] [Google Scholar] [Publisher Link]

[23] Yan-yan Song, and Ying Lu, "Decision Tree Methods: Applications for Classification and Prediction," *Shanghai Archives of Psychiatry*, vol. 27, no. 2, pp. 130-135, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[24] Mahajan Mayuri Vilas et al., "Detection of Phishing Website Using Machine Learning Approach," *2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques*, Mysuru, India, pp. 384-389, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[25] Vishakha Prashant Ratnaparkhi, and Sahil Siddharth Jambhulkar, "Framework for Detection and Prevention of Phishing Website Using Machine Learning Approach," *Computer Science*, 2020.

[26] Kiri Wagstaff, "Machine Learning that Matters," *arXiv*, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[27] Colin Whittaker, Brian Ryner, and Marria Nazif, "Large-Scale Automatic Classification of Phishing Pages," *Network and Distributed System Security (NDSS) Symposium*, pp. 1-14, 2010. [Google Scholar]

[28] Zhongheng Zhang, "Naïve Bayes Classification in R," *Annals of Translational Medicine*, vol. 4, no. 12, pp. 1-5, 2016. [CrossRef] [Google Scholar] [Publisher Link]