Phishing Website Detection

by

Aman Tripathi	**2021IMG-005**
Mithil Jogi	**2021IMG-033**
Muthiah Sivavelan	**2021IMG-034**
Yana Gupta	**2021IMG-064**

A report submitted for Artificial Intelligence Project

Bachelors Of Technology

In

IPG-MBA

ABV-INDIAN INSTITUTE OF INFORMATION

TECHNOLOGY AND MANAGEMENT GWALIOR-474015

Abstract

Phishing is regarded as one of the most severe types of online crime, and its prevalence has increased significantly in recent years. Phishing attacks can have a powerful negative impact on internet businesses. The goal of a Web phishing attack is to trick visitors into providing critical financial and personal information by fabricating a phishing website.

To address this issue, a number of traditional methods for identifying phishing websites have been proposed. Nevertheless, identifying phishing websites is a complex process because most of these tools are unable to dynamically determine if a newly discovered website is phishing or legitimate.

Phishing is a type of online fraud when a perpetrator sends phony messages that appear to be from a reliable source. The email contains a file or URL that, if clicked, can steal personal data or infect a machine with malware. Phishing efforts were traditionally executed using massive spam campaigns that arbitrarily targeted large populations of people. Getting as many individuals to open infected files or click on links was the aim. There are several methods for spotting this kind of attack. Using machine learning is one method. The machine learning model will receive the URLs that the user has submitted. The algorithm will then process the data and provide the results, indicating if the content is phishing or authentic. These URLs can be classified using a variety of machine learning methods, including SVM, Neural Networks, Random Forest, Decision Trees, XG boost, etc. The Random Forest Decision Tree classifiers are the subject of the suggested methodology.

Table of Contents

Chapters:

Introduction

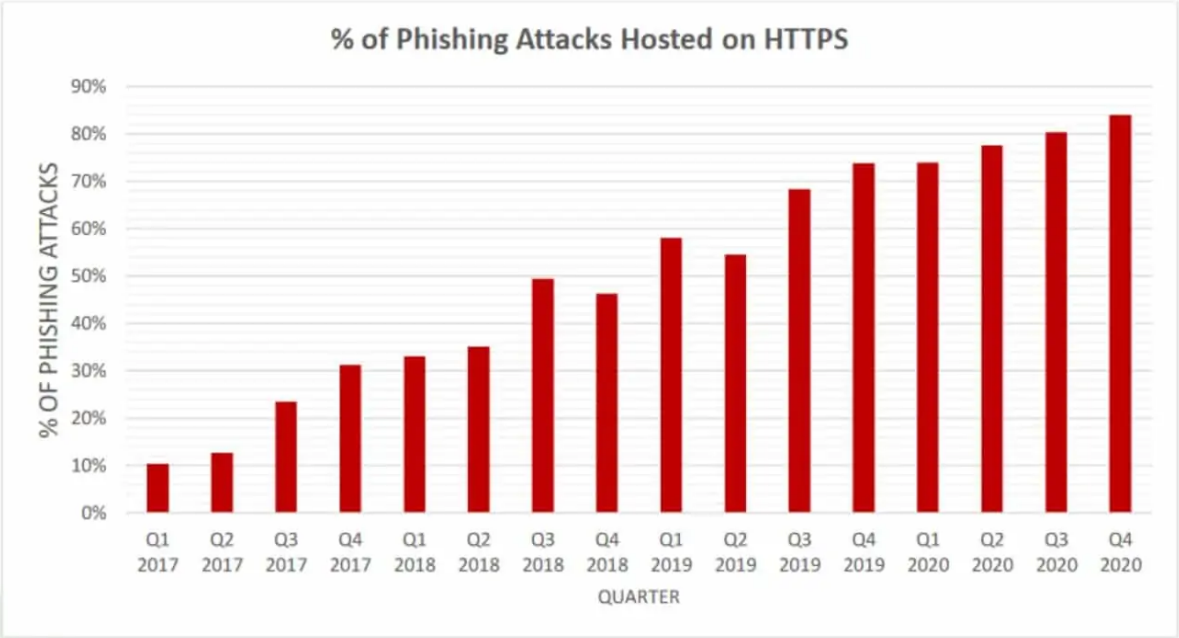
In recent years, the Internet has experienced growth due to a variety of services such as online business, entertainment, education, software downloads, and social networking. As a result, large amounts of data are constantly downloaded and sent to the internet. This gives thieves the opportunity to steal important personal or financial information such as usernames, passwords, account numbers and National Insurance numbers. This is called a phishing attack and is considered a major problem in cybersecurity.

1.1 Context:

In a phishing attack, an attacker creates a phishing website that resembles a legitimate website in order to trick Internet users into obtaining their financial and personal information. Phishing attacks are initiated by clicking on a link received in email. Victims receive an email with a link to update or verify their information. If the victim clicks on this link, his web browser will redirect him to a phishing website that looks like the original website. An attacker can steal sensitive information from Internet users who are asked to access sensitive information on phishing websites. Finally, attackers can steal money after a phishing attack.

Since phishing websites can target online businesses, banks, Internet users and governments, it is important to prevent phishing attacks early. However, detecting phishing websites is a difficult task as phishing attackers use many new methods to deceive Internet users.

The success of phishing website detection technology largely depends on correctly identifying phishing websites at the right time. Many traditional methods based on hard blacklists and whitelists have been proposed to identify phishing websites. However, this method is not efficient enough because new websites can be created in seconds. Therefore, most of the technology cannot detect and detect whether a new website is a phishing site or not. As a result, many new phishing websites will be classified as illegal.



As an alternative solution to traditional phishing website detection technology, some intelligent phishing website detection methods have been developed and proposed to effectively predict phishing websites. In recent years, intelligent phishing website detection solutions based on machine learning techniques have become widespread.

1.2 Implementation Workflow:

The implementation workflow followed during the implementation is as follows:

Step 1: Deciding the Problem.

Step 2: Exploring the data.

Step 3: Getting the data Ready.

Step 4: Picking and Fitting the model.

Step 5: Evaluating our model.

Step 6: Predicting on the custom URL.

1.3 Objective:

Our goals in this project are:

1. Get the dataset for url discovery.

2. Build an effective machine learning model that can classify and categorize given URLs.

3. Accurate detection and segmentation of URLs based on feature extraction.

4. Check different parts of the URL before guessing if it is a phishing site.

1.4 Research Result:

The model performs satisfactorily on testing data and is able to segment the URL samples correctly with approx 97.4% accuracy. Detailed results have been discussed further.

Problem statements

- Internet growth fuels a surge in phishing: Increasing online activity has led to a rise in deceptive phishing sites.
- Deception and data disclosure: Phishing sites pretend to be real and trick users into revealing sensitive data.
- Consequences: Phishing results in financial losses, identity theft, and reputation damage.
- Real-time detection challenge: There's a need for an accurate system to spot phishing sites promptly.
- Evolving tactics: Current methods struggle to keep up with cybercriminals, demanding more advanced detection solutions.

Data Sourcing

At the Phishing Website Detection Project, we rely on Kaggle as a valuable source of data in the field of cybersecurity. Kaggle is a well-known platform that hosts a wide range of datasets and resources related to various domains, including cybersecurity and machine learning. We utilize data from Kaggle to provide you with accurate and up-to-date information on the detection of phishing websites.

Our commitment to data quality, proper attribution, and data privacy ensures that the information we provide is reliable and secure, in compliance with Kaggle's terms and policies.

Link for the Dataset.

Exploratory Data Analysis (EDA)

3.1 Data Set:

Data Collection: Obtained the dataset, from Kaggle. For this project, we collected a dataset of URLs, including both legitimate and phishing URLs. Kaggle happens to be one of the free sources and a collaborative platform for ML enthusiasts.

1. *Loading the Dataset:*

```
In [3]:
#load dataset

data = pd.read_csv("./phishing.csv")
data.head()
```

Out[98]:

	Index	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTTPS	DomainRegLen	...	UsingPopupl
0	0	1	1	1	1	1	-1	0	1	-1	...	
1	1	1	0	1	1	1	-1	-1	-1	-1	...	
2	2	1	0	1	1	1	-1	-1	-1	1	...	
3	3	1	0	-1	1	1	-1	1	1	-1	...	
4	4	-1	0	-1	1	-1	-1	1	1	-1	...	

5 rows × 32 columns

3.2 Workflow:

2. *Number of Rows and Columns in Dataset: There are 11054 rows and 32 columns in the dataset.*

```
In [99]:
data.shape

Out[99]: (11054, 32)
```

3. *Getting the columns: Here the title of each column in the dataset is being displayed.*

```
In [100…  data.columns

Out[100…  Index(['Index', 'UsingIP', 'LongURL', 'ShortURL', 'Symbol@', 'Redirecting//',
                 'PrefixSuffix-', 'SubDomains', 'HTTPS', 'DomainRegLen', 'Favicon',
                 'NonStdPort', 'HTTPSDomainURL', 'RequestURL', 'AnchorURL',
                 'LinksInScriptTags', 'ServerFormHandler', 'InfoEmail', 'AbnormalURL',
                 'WebsiteForwarding', 'StatusBarCust', 'DisableRightClick',
                 'UsingPopupWindow', 'IframeRedirection', 'AgeofDomain', 'DNSRecording',
                 'WebsiteTraffic', 'PageRank', 'GoogleIndex', 'LinksPointingToPage',
                 'StatsReport', 'class'],
                dtype='object')
```

4. *Summary of a DataFrame's composition, including data types, non-null values, and memory usage.*

```
In [101…  data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11054 entries, 0 to 11053
Data columns (total 32 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Index                11054 non-null  int64
 1   UsingIP              11054 non-null  int64
 2   LongURL              11054 non-null  int64
 3   ShortURL             11054 non-null  int64
 4   Symbol@              11054 non-null  int64
 5   Redirecting//        11054 non-null  int64
 6   PrefixSuffix-        11054 non-null  int64
 7   SubDomains           11054 non-null  int64
 8   HTTPS                11054 non-null  int64
 9   DomainRegLen         11054 non-null  int64
 10  Favicon              11054 non-null  int64
 11  NonStdPort           11054 non-null  int64
 12  HTTPSDomainURL       11054 non-null  int64
 13  RequestURL           11054 non-null  int64
 14  AnchorURL            11054 non-null  int64
 15  LinksInScriptTags    11054 non-null  int64
 16  ServerFormHandler    11054 non-null  int64
 17  InfoEmail            11054 non-null  int64
 18  AbnormalURL          11054 non-null  int64
 19  WebsiteForwarding    11054 non-null  int64
 20  StatusBarCust        11054 non-null  int64
 21  DisableRightClick    11054 non-null  int64
 22  UsingPopupWindow     11054 non-null  int64
 23  IframeRedirection    11054 non-null  int64
 24  AgeofDomain          11054 non-null  int64
 25  DNSRecording         11054 non-null  int64
 26  WebsiteTraffic       11054 non-null  int64
 27  PageRank             11054 non-null  int64
 28  GoogleIndex          11054 non-null  int64
 29  LinksPointingToPage  11054 non-null  int64
 30  StatsReport          11054 non-null  int64
 31  class                11054 non-null  int64
dtypes: int64(32)
memory usage: 2.7 MB
```

5. *Counting the unique values in each column: The count of unique values in each column of the DataFrame*.

```
In [102…  data.nunique()

Out[102…  Index                 11054
          UsingIP                   2
          LongURL                   3
          ShortURL                  2
          Symbol@                   2
          Redirecting//             2
          PrefixSuffix-             2
          SubDomains                3
          HTTPS                     3
          DomainRegLen              2
          Favicon                   2
          NonStdPort                2
          HTTPSDomainURL            2
          RequestURL                2
          AnchorURL                 3
          LinksInScriptTags         3
          ServerFormHandler         3
          InfoEmail                 2
          AbnormalURL               2
          WebsiteForwarding         2
          StatusBarCust             2
          DisableRightClick         2
          UsingPopupWindow          2
          IframeRedirection         2
          AgeofDomain               2
          DNSRecording              2
          WebsiteTraffic            3
          PageRank                  2
          GoogleIndex               2
          LinksPointingToPage       3
          StatsReport               2
          class                     2
          dtype: int64
```

6. *Dropping the column*: The 'Index' column was dropped from the dataset as it provided no meaningful information or utility for the analysis and modeling tasks. Its values were merely arbitrary row identifiers and, therefore, were not necessary for the study's objectives. By removing this column, the dataset was made more concise and suitable for the analysis.

```
In [103…  data = data.drop(['Index'],axis = 1)
```

7. *Describing the data after dropping the column 'Index'*

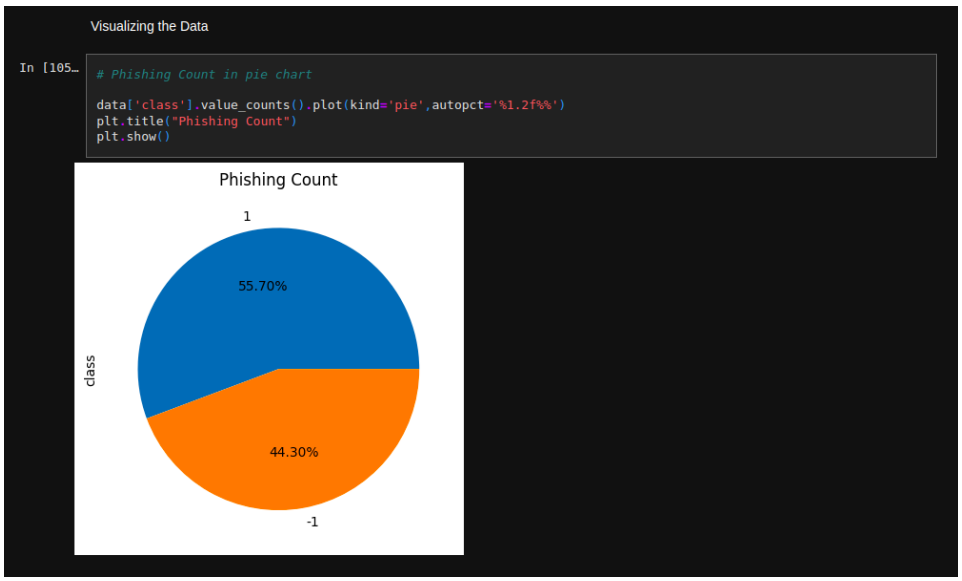
```
In [104…  data.describe().T
```

RESULT:

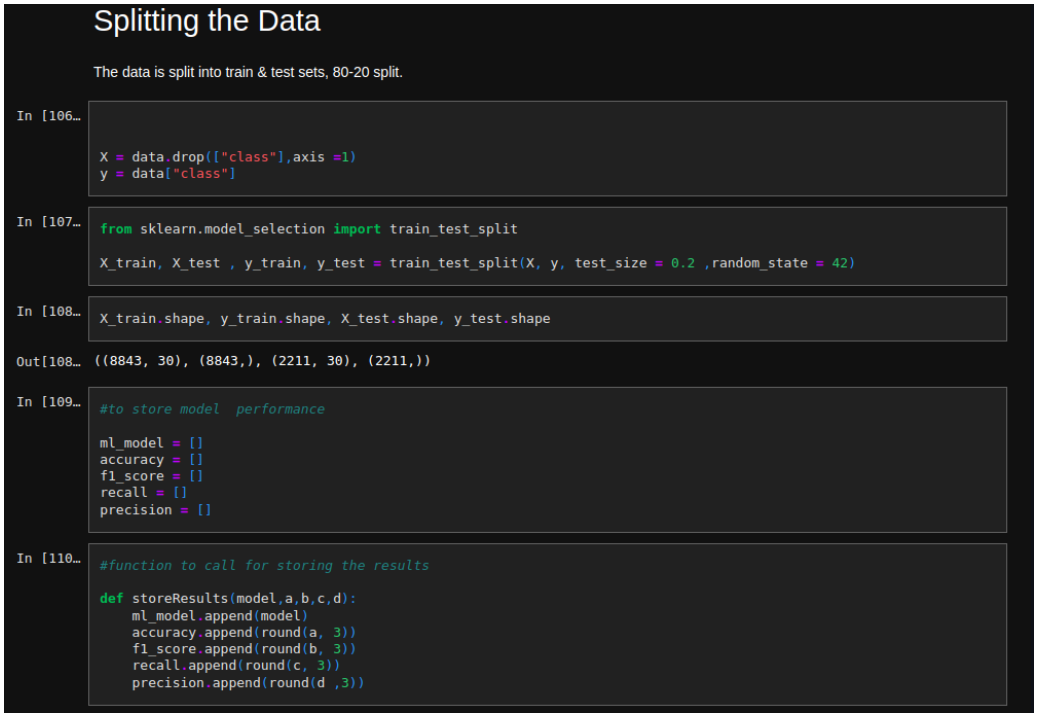
Out[104…

	count	mean	std	min	25%	50%	75%	max
UsingIP	11054.0	0.313914	0.949495	-1.0	-1.0	1.0	1.0	1.0
LongURL	11054.0	-0.633345	0.765973	-1.0	-1.0	-1.0	-1.0	1.0
ShortURL	11054.0	0.738737	0.674024	-1.0	1.0	1.0	1.0	1.0
Symbol@	11054.0	0.700561	0.713625	-1.0	1.0	1.0	1.0	1.0
Redirecting//	11054.0	0.741632	0.670837	-1.0	1.0	1.0	1.0	1.0
PrefixSuffix-	11054.0	-0.734938	0.678165	-1.0	-1.0	-1.0	-1.0	1.0
SubDomains	11054.0	0.064049	0.817492	-1.0	-1.0	0.0	1.0	1.0
HTTPS	11054.0	0.251040	0.911856	-1.0	-1.0	1.0	1.0	1.0
DomainRegLen	11054.0	-0.336711	0.941651	-1.0	-1.0	-1.0	1.0	1.0
Favicon	11054.0	0.628551	0.777804	-1.0	1.0	1.0	1.0	1.0
NonStdPort	11054.0	0.728243	0.685350	-1.0	1.0	1.0	1.0	1.0
HTTPSDomainURL	11054.0	0.675231	0.737640	-1.0	1.0	1.0	1.0	1.0
RequestURL	11054.0	0.186720	0.982458	-1.0	-1.0	1.0	1.0	1.0
AnchorURL	11054.0	-0.076443	0.715116	-1.0	-1.0	0.0	0.0	1.0
LinksInScriptTags	11054.0	-0.118238	0.763933	-1.0	-1.0	0.0	0.0	1.0
ServerFormHandler	11054.0	-0.595712	0.759168	-1.0	-1.0	-1.0	-1.0	1.0
InfoEmail	11054.0	0.635788	0.771899	-1.0	1.0	1.0	1.0	1.0
AbnormalURL	11054.0	0.705446	0.708796	-1.0	1.0	1.0	1.0	1.0
WebsiteForwarding	11054.0	0.115705	0.319885	0.0	0.0	0.0	0.0	1.0
StatusBarCust	11054.0	0.762077	0.647516	-1.0	1.0	1.0	1.0	1.0
DisableRightClick	11054.0	0.913877	0.406009	-1.0	1.0	1.0	1.0	1.0
UsingPopupWindow	11054.0	0.613353	0.789845	-1.0	1.0	1.0	1.0	1.0
IframeRedirection	11054.0	0.816899	0.576807	-1.0	1.0	1.0	1.0	1.0
AgeofDomain	11054.0	0.061335	0.998162	-1.0	-1.0	1.0	1.0	1.0
DNSRecording	11054.0	0.377239	0.926158	-1.0	-1.0	1.0	1.0	1.0
WebsiteTraffic	11054.0	0.287407	0.827680	-1.0	0.0	1.0	1.0	1.0
PageRank	11054.0	-0.483626	0.875314	-1.0	-1.0	-1.0	1.0	1.0
GoogleIndex	11054.0	0.721549	0.692395	-1.0	1.0	1.0	1.0	1.0
LinksPointingToPage	11054.0	0.343948	0.569936	-1.0	0.0	0.0	1.0	1.0
StatsReport	11054.0	0.719739	0.694276	-1.0	1.0	1.0	1.0	1.0
class	11054.0	0.113986	0.993527	-1.0	-1.0	1.0	1.0	1.0

8. *Visualizing the data*: The value 1 represents the website is a Phishing Website and the value -1 represents the website is not a Phishing Website. So in the dataset used 55.70% websites are Phishing websites and 44.30% websites are non-phishing websites.



9. *Splitting the Data*: The data is split into 80-20, where 80% data is used for training and 20% is used to test.



Data Cleaning

Data cleaning, also known as data cleansing, is a crucial step in the data preparation process that involves identifying and rectifying errors, inconsistencies, and inaccuracies in a dataset. It is a fundamental aspect of data quality management and is essential for ensuring that data is reliable, accurate, and ready for analysis or other data-driven tasks.

Dropping the Index Column:

Model Building

Model Selection: In the model selection process, accuracy played a pivotal role as the primary performance metric. The choice of the most suitable model was based on its ability to make accurate predictions on the test data.

Selected Models:

- K-Nearest Neighbour (KNN): KNN is a simple yet effective algorithm for classification tasks. It operates on the principle that similar data points tend to have similar outcomes.
- Decision Tree: Decision trees are versatile models used for classification and regression. They represent decision-making processes through a tree-like structure.
- Random Forest: Random Forest is a powerful ensemble learning algorithm that combines multiple decision trees to make more accurate predictions.
- Gradient Boosting: Gradient Boosting is a boosting algorithm that combines weak learners into strong learners, making it effective for handling complex data relationships.

Derived metrics

Derived metrics create a new variable from the existing variable to get insightful information from the data by analyzing the data.it includes:- feature binning , feature encoding , from domain knowledge , calculated from data.

1. **Feature Binning** :

   Feature Binning converts or transforms continuous variable to categorical variable. It can also be used to identify missing values or outliers.

2. **Feature encoding** :

   A. Label encoding : Label encoding is a technique to transform categorical variables into numerical by assigning a numerical value to each of the categories
   B. Target encoding : In target encoding , we calculate the average of the dependent variable for each category and replace the category variable with the mean value.
   C. One-hot encoding : This technique is used when independent variables are nominal . It creates k different columns each for a category and replaces one column with the rest of the columns is 0.
   D. Hash encoder: The Hash encoder represents the categorical independent variable using the new dimensions . Here the user can fix the number of dimensions after the transformative using component argument.

3. Performance Statistical Terms:

3.1 Accuracy:

It measures the overall correctness of a classification model. It calculates the ratio which is equal to correctly predicted instances divided by the the total number of instances in the dataset. High accuracy indicates that the model makes fewer mistakes.

3.2 Precision:

It is a measure of how many of the positive predictions made by a model are actually correct. It calculates the ratio which is equal to true positives (correctly predicted positive instances) divided by the total number of positive predictions. High precision tells low false positive rate.

3.3 Recall:

It is also known as sensitivity or true positive rate, measures how many of the actual positive instances the model correctly predicted. It calculates the ratio which is equal to true positives divided by the total number of actual positive instances. High recall tells a low false negative rate.

3.4 F1 Score:

Metric which balances precision and recall. Harmonic mean of these two values. The F1 score is particularly useful when you want to find a balance between making accurate positive predictions (precision) and capturing all actual positive instances (recall). It is especially valuable when dealing with imbalanced datasets.

In summary, accuracy assesses overall correctness, precision focuses on the accuracy of positive predictions, recall emphasizes the model's ability to capture actual positive instances, and the F1 score combines both precision and recall which provide a balanced evaluation of a classification model's performance. These metrics are commonly used to evaluate the effectiveness of machine learning models, particularly in tasks like binary classification.

Supervised Machine Learning:

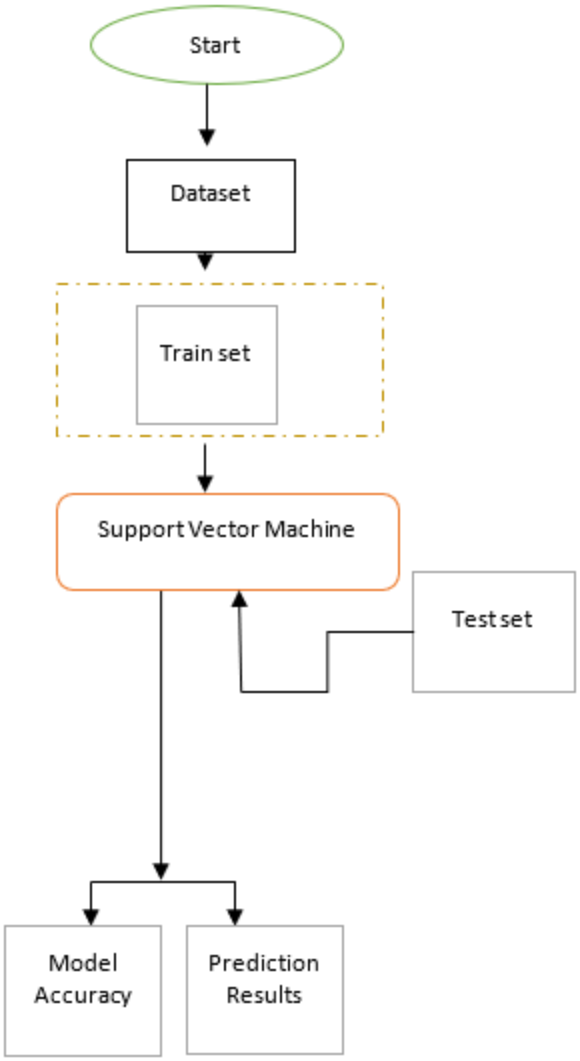
Supervised machine learning is a type of machine learning where the algorithm is trained on a labeled dataset, which means that each example in the training data is paired with the correct output. The goal of supervised learning is to learn a mapping from input data to the corresponding output based on the patterns and relationships in the training data. In other words, the algorithm learns to make predictions or decisions based on the input data by generalizing from the labeled examples it has seen.

Here are some key characteristics of supervised machine learning:

1. Labeled Data: The training dataset consists of input-output pairs where the output (or target) is known. For example, in a classification task, the input data could be images of animals, and the output labels could be the names of the animals (e.g., "cat," "dog," "horse").

2. Training Phase: During the training phase, the machine learning model tries to find a mapping or function that can predict the correct output for a given input. This is often done by adjusting the model's parameters to minimize the difference between its predictions and the true output in the training data.

3. Testing and Inference: Once the model is trained, it can be used to make predictions on new, unseen data. The model's performance is typically evaluated on a separate dataset (test set) to assess how well it generalizes to new examples. This evaluation helps determine the model's accuracy and effectiveness.



4.1 TYPES:

1. **DECISION TREE:**

A decision tree is a versatile machine learning algorithm used for classification and regression tasks. It represents a decision-making process through a tree-like structure. At the root node, the entire dataset is considered. The algorithm selects features that best split the data based on information gain or impurity reduction. Data is then divided into subsets, creating branches in the tree. This

process recursively continues, with feature selection and splitting until stopping criteria, such as a maximum depth or minimum samples per leaf, are met.

Leaf nodes, the terminal points of the tree, represent the final decision. In classification, each leaf node signifies a class label, while in regression, it represents a predicted value. To make predictions for new data, you follow the tree from the root node down the branches, making decisions based on feature values until a leaf node is reached, which provides the final prediction.

Decision trees are valued for their interpretability and ability to handle various data types, including both categorical and numerical features. Nevertheless, they can overfit by creating overly complex trees that fit training data too closely. Techniques like pruning and ensemble methods like Random Forests can mitigate overfitting and enhance decision tree performance. Overall, decision trees are a fundamental tool in machine learning, finding applications in diverse fields such as finance, healthcare, and recommendation systems.

ALGORITHM:

```
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

# fit the model
tree.fit(X_train, y_train)
```

```
#predicting the target value from the model for the samples

y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)
```

```
training_accuracy = []
test_accuracy = []

depth = range(1,30)
for n in depth:
    tree_test = DecisionTreeClassifier(max_depth=n)

    tree_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(tree_test.score(X_train, y_train))
    # record generalization accuracy


    test_accuracy.append(tree_test.score(X_test, y_test))
```

```
#plotting the training & testing accuracy for max_depth from 1 to 30
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
```

```
training_accuracy = []
test_accuracy = []

depth = range(1,30)
for n in depth:
    tree_test = DecisionTreeClassifier(max_depth=n)

    tree_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(tree_test.score(X_train, y_train))
    # record generalization accuracy


    test_accuracy.append(tree_test.score(X_test, y_test))
```

```
#plotting the training & testing accuracy for max_depth from 1 to 30
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
```

2. **RANDOM FOREST:**

Random Forest is a powerful ensemble learning algorithm in machine learning that leverages multiple decision trees to make more accurate and robust predictions. It combines the strengths of individual decision trees while mitigating their weaknesses, such as overfitting.

In a Random Forest, a predefined number of decision trees are created. Each tree is trained on a bootstrapped subset of the training data (a random sample with replacement). Additionally, during the creation of each tree, a random subset of features is considered for splitting at each node. This randomness injects diversity into the forest, making the individual trees different from one another.

When it's time to make a prediction, each tree in the Random Forest provides its own prediction (classification or regression). In classification tasks, the final prediction is often determined by a majority vote among the individual trees, while in regression tasks, it's the average of their predictions.

Random Forests offer several advantages, including high accuracy, robustness against overfitting, and the ability to handle large and complex datasets. They are also capable of feature importance ranking, which helps identify the most influential features in the data.

ALGORITHM:

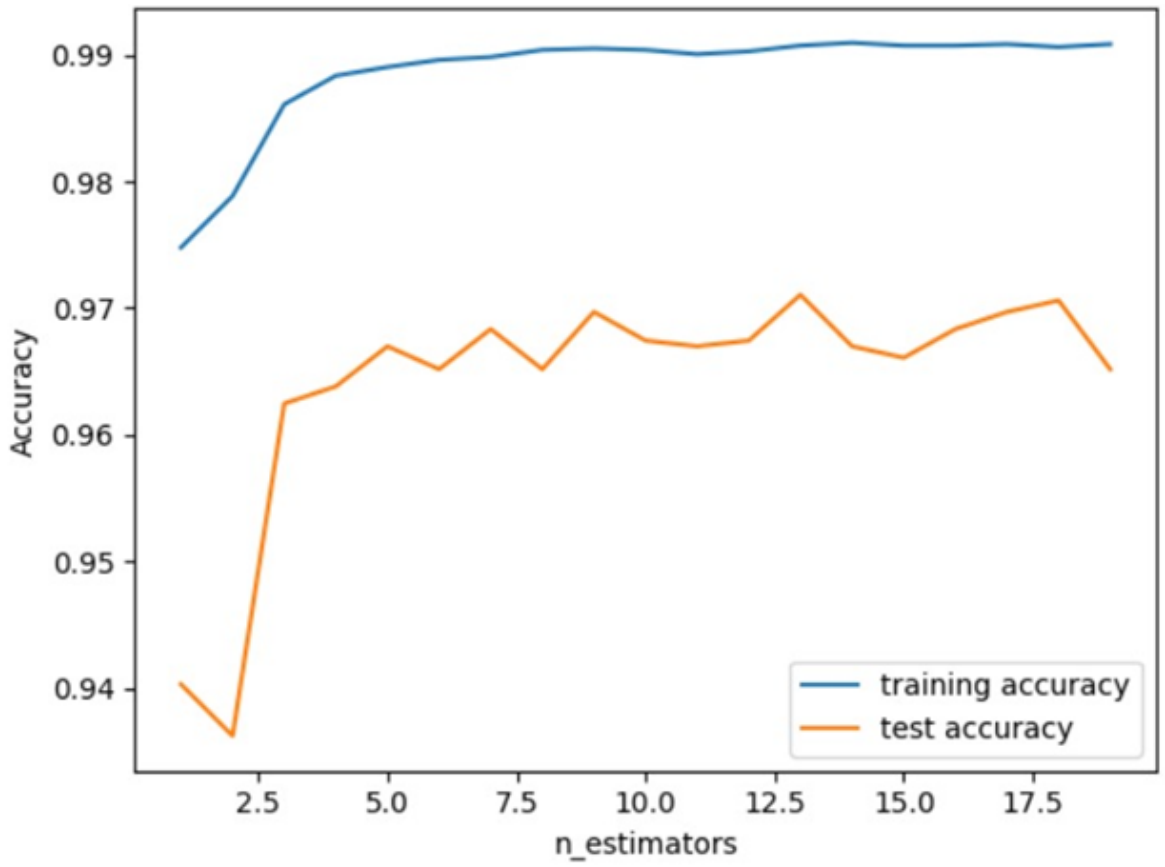
```
# K-Nearest Neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)

# fit the model
knn.fit(X_train,y_train)
```

```
X_train = X_train.values
X_test = X_test.values
```

```
#predicting the target value from the model for the samples
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
```



3. **KNN:**

K-Nearest Neighbours (KNN) is a simple yet effective supervised machine learning algorithm used for classification and regression tasks. It operates based on the principle that similar data points tend to have similar outcomes. KNN is a non-parametric, instance-based algorithm, meaning it doesn't make underlying assumptions about the data distribution and relies on the training data directly for making predictions.

In the Training phase KNN stores the entire dataset in memory, including both the feature values and their corresponding class labels (in the case of classification) or target values (in the case of regression). When making a prediction for a new data point, KNN identifies the k-nearest data points (neighbors) in the training dataset to the new point. The "k" value is a hyperparameter set by the user. KNN assigns the class label that is most frequently represented among the k-nearest neighbors to the new data point. In a regression task, KNN computes the average (or weighted average) of the target values of the k-nearest neighbors as the predicted value for the new data point.

KNN's performance heavily depends on the choice of the hyperparameter "k" and the distance metric used to measure similarity between data points (commonly Euclidean distance). Smaller values of "k" may lead to noisy predictions, while larger values can over smooth the decision boundaries.

KNN is easy to understand and implement, making it suitable for small to medium-sized datasets. However, it can be computationally expensive, especially for large datasets, as it requires searching through the entire training dataset for each prediction. Preprocessing and feature scaling are often necessary to improve KNN's performance.

ALGORITHM:

```
# K-Nearest Neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)

# fit the model
knn.fit(X_train,y_train)
```

```
X_train = X_train.values
X_test = X_test.values
```

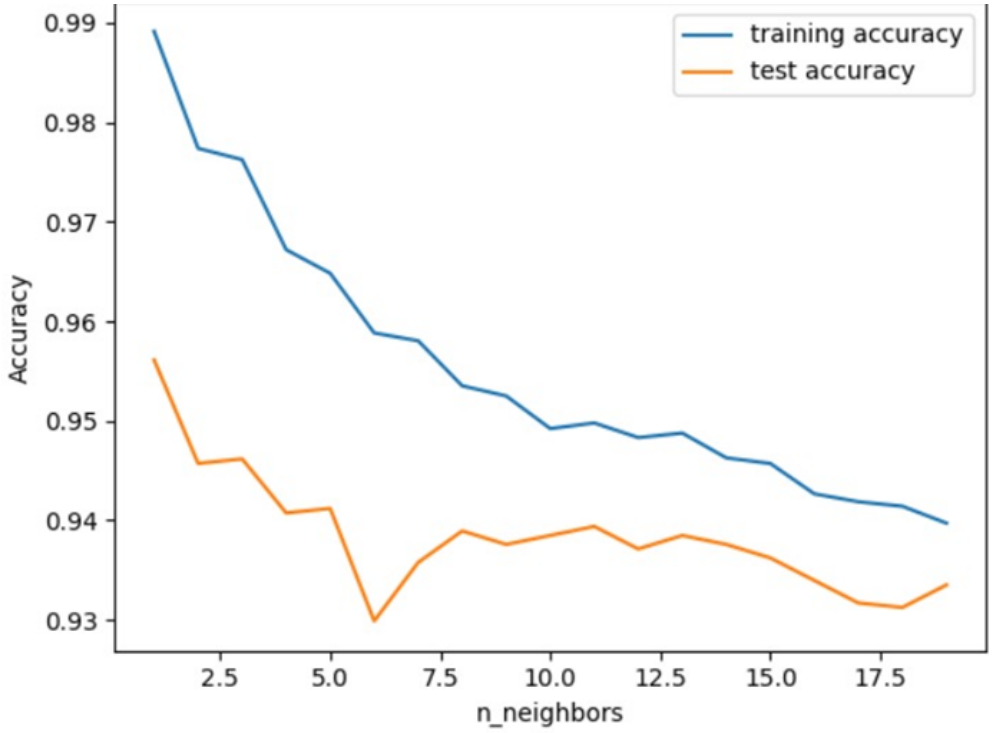
```
#predicting the target value from the model for the samples
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
```

```
training_accuracy = []
test_accuracy = []
depth = range(1,20)
for n in depth:
    knn = KNeighborsClassifier(n_neighbors=n)

    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(knn.score(X_test, y_test))



#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend();
```



4. **GRADIENT BOOST**:

Gradient Boosting is a powerful boosting algorithm that combines several weak learners into strong learners, in which each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using gradient descent. In each iteration, the algorithm computes the gradient of the loss function with respect to the predictions of the current ensemble and then trains a new weak model to minimize this gradient. The predictions of the new model are then added to the ensemble, and the process is repeated until a stopping criterion is met.

The process involves data preparation, model selection from libraries like XGBoost or LightGBM, and hyperparameter tuning through techniques like cross-validation. Training the model is crucial, with each tree correcting errors from previous ones, while model performance is evaluated based on classification or regression metrics. Feature importance analysis helps identify significant features in the model. The model is deployed for real-world predictions through APIs. Continuous monitoring and updates with new data are essential to maintain accuracy. Gradient Boosting is favored for its ability to handle complex data relationships and deliver accurate predictions, but hyperparameter tuning is vital to prevent overfitting.

INPUT:

```
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

# fit the model
forest.fit(X_train,y_train)
```

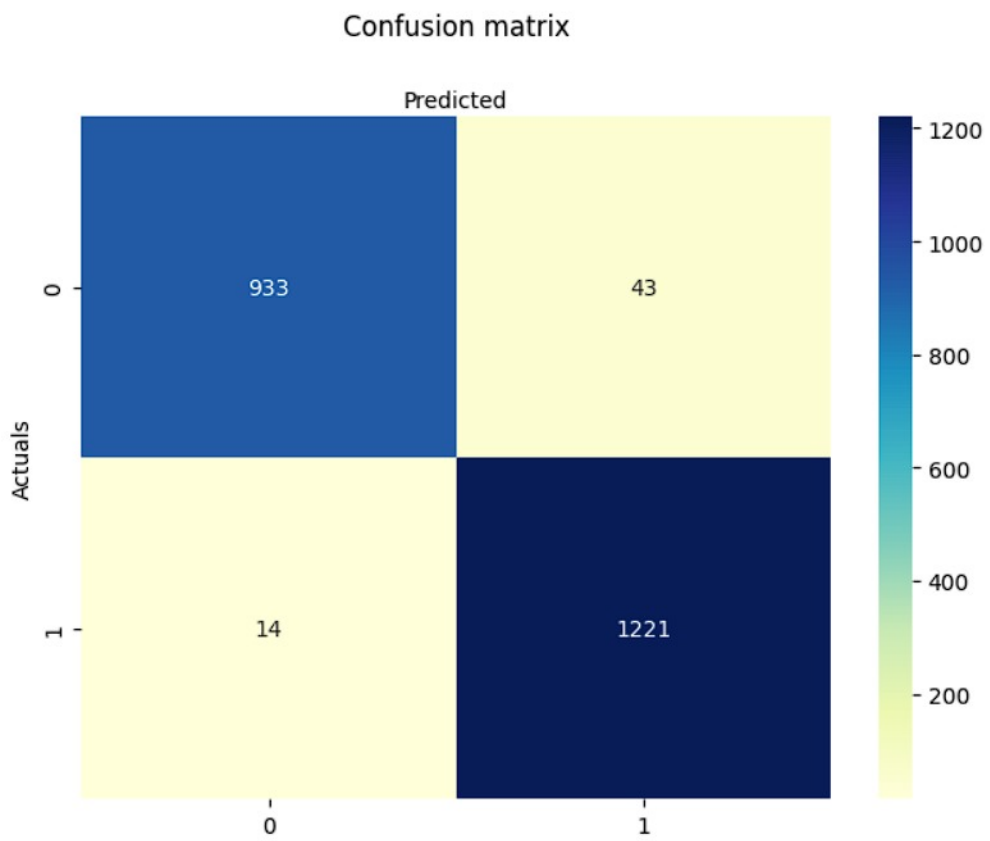
```
y_train_forest = forest.predict(X_train)
y_test_forest = forest.predict(X_test)
```

ALGORITHM:

```
training_accuracy = []
test_accuracy = []
depth = range(1,20)
for n in depth:
    forest_test =  RandomForestClassifier(n_estimators=n)

    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))


#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_estimators")
plt.legend();
```



Discussion and Conclusion

Output:

```
url="https://github.com"
#can provide any URL. this URL was taken from PhishTank
obj = FeatureExtraction(url)
x = np.array(obj.getFeaturesList()).reshape(1,30)
y_pred =knn.predict(x)[0]
if y_pred==1:
  print("We guess it is a safe website")
else:
  print("Caution! Suspicious website detected")
```

```
We guess it is a safe website
```

Conclusion:

In conclusion, the results of our phishing website detection project are indeed promising, with the highest achieved accuracy rate of 97.4% by using Gradient Boost Model. This remarkable accuracy demonstrates the effectiveness of our detection model in distinguishing between legitimate and fraudulent websites. Our efforts in developing advanced algorithms and employing cutting-edge technology have paid off, contributing significantly to the enhancement of online security.

While our achievement in accuracy is commendable, it is important to acknowledge that the battle against phishing attacks is an ongoing one, with cybercriminals continuously evolving their tactics. Therefore, our work does not end here. We must remain vigilant, consistently update our models, and adapt to emerging threats to maintain the highest level of protection for users.

This project represents a crucial step forward in the field of cybersecurity, offering a robust defense against phishing threats. The 97.4% accuracy rate serves as a testament to our commitment to safeguarding online users from malicious intent. We hope that this research will inspire further innovations and collaborations in the fight against cybercrime, ultimately creating a safer digital environment for all.

References:

Data Files

This folder has the raw & extracted datafiles of this project. The description of each file is as follows:

1.Benign_list_big_final.csv: (https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/1.Benign_list_big_final.csv):- This file has list of legitimate urls. The total count is 35,300. The source of the dataset is University of New Brunswick, https://www.unb.ca/cic/datasets/url-2016.html.

2.online-valid.csv: (https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/2.online-valid.csv):- This file is downloaded from the open source service called PhishTank. This service provides a set of phishing URLs in multiple formats like csv, json etc. that gets updated hourly. To download the latest data: https://www.phishtank.com/developer_info.php.

3.legitimate.csv: (https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/3.legitimate.csv) :- This file has the

extracted features of the 5000 legitimate URLs which are randomly selected from the '1.Benign_list_big_final.csv' file.

4.phishing.csv (https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/4.phishing.csv):- This file has the extracted features of the 5000 phishing URLs which are randomly selected from the '2.online-valid.csv' file.

5.urldata.csv (https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/5.urldata.csv):- This file is nothing but a combination of the above two files. It contains extracted features of 10,000 URLs both legitimate & phishing.