WEBSITE PHISHING DETECTION

using machine learning techniques

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the

Requirement for the award of the

Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

SPECIALIZATION IN

ARTIFICIAL INTELLIGENCE

by

K Shiva Kalyan Kumar(19BCI7076)

Under the Guidance of

Dr. E. Ajith Jubilson

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT-AP UNIVERSITY

AMARAVATI- 522237

January 2023

CERTIFICATE

This is to certify that the Capstone Project work titled "**WEBSITE PHISHING DETECTION**" that is being submitted by **K Shiva Kalyan Kumar (19BCI7076)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma, and the same is certified.

Dr. E. Ajith Jubilson

Guide

The thesis is satisfactory/unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR DEAN

B. Tech. CSE School Of Computer Science and Engineering

ACKNOWLEDGMENTS

I would like to express my gratitude to **Dr. G. Viswanathan**, Chancellor**, Dr. Sekar Viswanathan**, Vice president, **Dr. S. V. Kota Reddy**, Vice Chancellor, and Dean **Dr. Sudha S V**, School of Computer Science and Engineering, for providing a conducive environment to work during the tenure of the course.

In a jubilant mood, I express ingeniously my whole-hearted thanks to **Dr. B. Srinivasa Rao**, the Program Chair and Professor, at the School of Computer Science and Engineering Specialization in Artificial Intelligence and it's my pleasure to express a deep sense of gratitude to **Dr. E. Ajith Jubilson**, Associate Professor, School of Computer Science and Engineering, VIT-AP, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor.

It is indeed my pleasure to thank all faculty members working as limbs of our university for their selfless enthusiasm and timely encouragement showered on me with zeal, prompting the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

Last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 01–01–2023 K. Shiva Kalyan Kumar

ABSTRACT

Phishing is the simplest method for obtaining sensitive information from consumers who are not expecting it. Phishing attempts to steal confidential information like usernames, passwords, and bank account details. The cyber security platform is currently looking for dependable and consistent phishing website detection methods. This project uses logistic Regression, DT, SVM, Random Forests, KNN, Neural Networks, Naïve Bayes, and Genetic Algorithms to detect phishing websites. This is done by extracting and evaluating numerous aspects of both authentic and phishing URLs. The project's objective is to compare accuracy rates to find the best ML algorithm and identify phishing URLs.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Nowadays, phishing is becoming the main concern for security researchers because of the ease of creating a fake website like the original one. Experts can identify fake websites but the rest of the people cannot identify them and become the victims of phishing attacks. In several severe cases, clients are exploited for their personal and critical data. Top 2 Phishing Scams and Data Breaching of all time.

The Belgian Crelan Bank was the target of a BEC fraud that resulted in a loss of $75.8 million for the company. The phisher gets into the account of a high-ranking executive at a company and tells his or her coworkers to send money to an attacker's account. An internal audit revealed the Crelan Bank phishing attempt, and the company was able to absorb the loss due to its substantial internal reserves.

Additionally, the Austrian aerospace parts manufacturer FACC lost a significant amount of finances to a BEC scam. In 2016, the company revealed that an employee in the accounting department was instructed by a phisher who impersonated the company's CEO to transfer $61 million to a bank account controlled by the attacker. This was unusual because the company decided to fire its CFO and CEO and sue them. The company sought $11 million in damages from the two executives for failing to implement internal supervision and security controls that could have fended off the attack. This lawsuit showed executives the personal risk of not doing "due diligence" on cybersecurity.

The Marriott International data leak impacted nearly 500 million records. The data that was exposed included credit card numbers, expiration dates, travel information, contact information, passport numbers, Starwood Preferred Guest credentials, and other sensitive information. As a result of class-action lawsuits and fines imposed by the United Kingdom, the company lost $24 million.

412.2 million people were affected by the Adult Friend Finder Networks data breach. The stolen data included passwords, email addresses, and names. The disclosure of sensitive account information.

So this leads to the point that we need a solution to curb these phishing attacks. And this research is done for the same. The models proposed in this research are Logistic Regression, DT, SVM, Random Forests, KNN, Neural Networks, Naïve Bayes, and Genetic Algorithms.

1.1 BACKGROUND AND LITERATURE REVIEW

A. AI meta-learners and Extra Trees

In their research, Yazan [1] used the Extra Tree Algorithm and AI Meta - Learners to identify phishing websites. The following methods are suggested by the study: BET, LBET, RoFET, and others The four steps of the research method are depicted in Figure 1. Pre-processing begins with the data obtained from the UCI Phishing Website Dataset. Second, the proposed methods are parameterized and the Meta-Learner and Base Learner Algorithms are initialized. Last but not least, the combination of Base Learners and Meta-Learners. An n-fold cross-validation algorithm where n = 10 is being used to create the AI-based Meta-Learner models. The proposed approaches are then evaluated. The AdaBoost and ABET are combined in one method. The Extra Tree algorithm and the M1 meta-learner PCF filter with low bias and high accuracy. BET is a meta-learner that uses a dataset to combine 150 Extra-Tree results. The dataset's noise and outliers are addressed by LBET, which combines the Extra-Tree algorithm with the LogitBoost meta-learner. The accuracy and performance of the models proposed are as follows: ABET had a percentage of 97.48 percent, RoFET had a percentage of 97.449 percent, BET had a percentage of 97.44%, and LBET had a percentage of 97.576%.

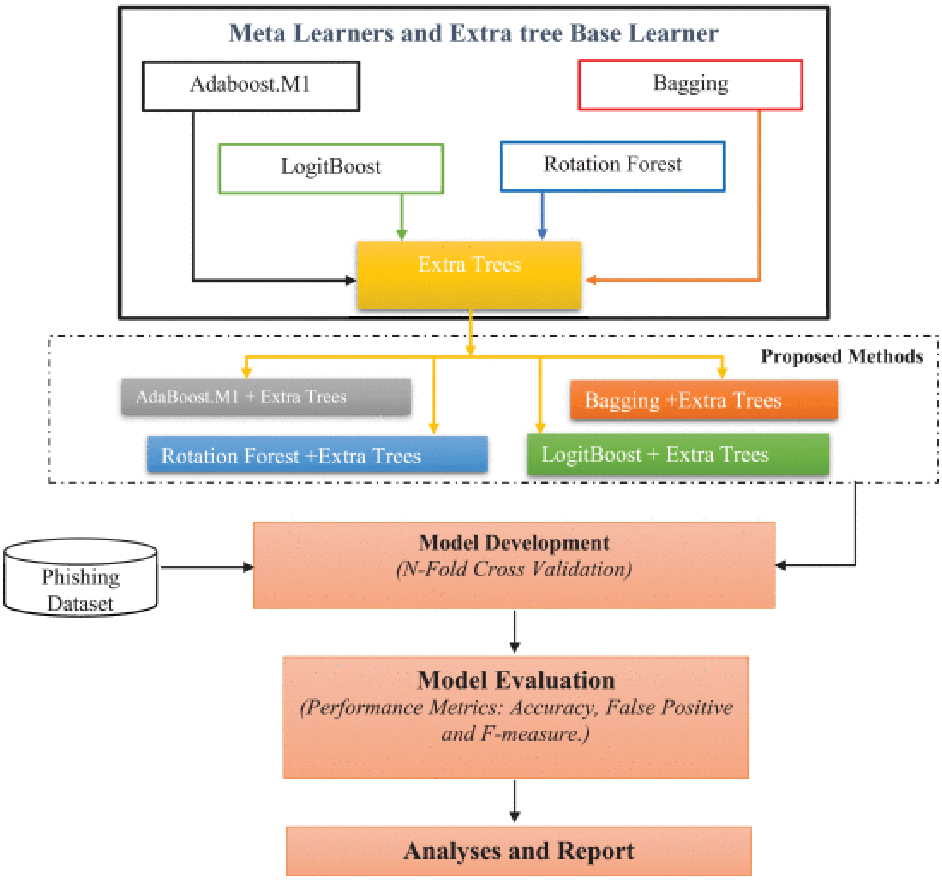

Figure 1: Experimental flowchart of Meta Learners

B. PhishSim: Aiding Phishing Website detection using the feature-free tool

Using the Normalized Compression Distance (NCD), Rizka [2] suggested a feature-free approach for identifying phishing websites. The similarities between two web pages are compressed by this parameter-free similarity measure. The method used in this paper is shown in Figure 2. Any requirement for feature extraction is eliminated by this. To begin, they extracted phishing prototypes using the Furthest Point First algorithm to select instances that represented the cluster of phishing websites. When idea drift occurred, they used incremental learning as a foundation for continuous and adaptive detection without extracting new characteristics. The proposed model's exhibition/precision is 96.59%.

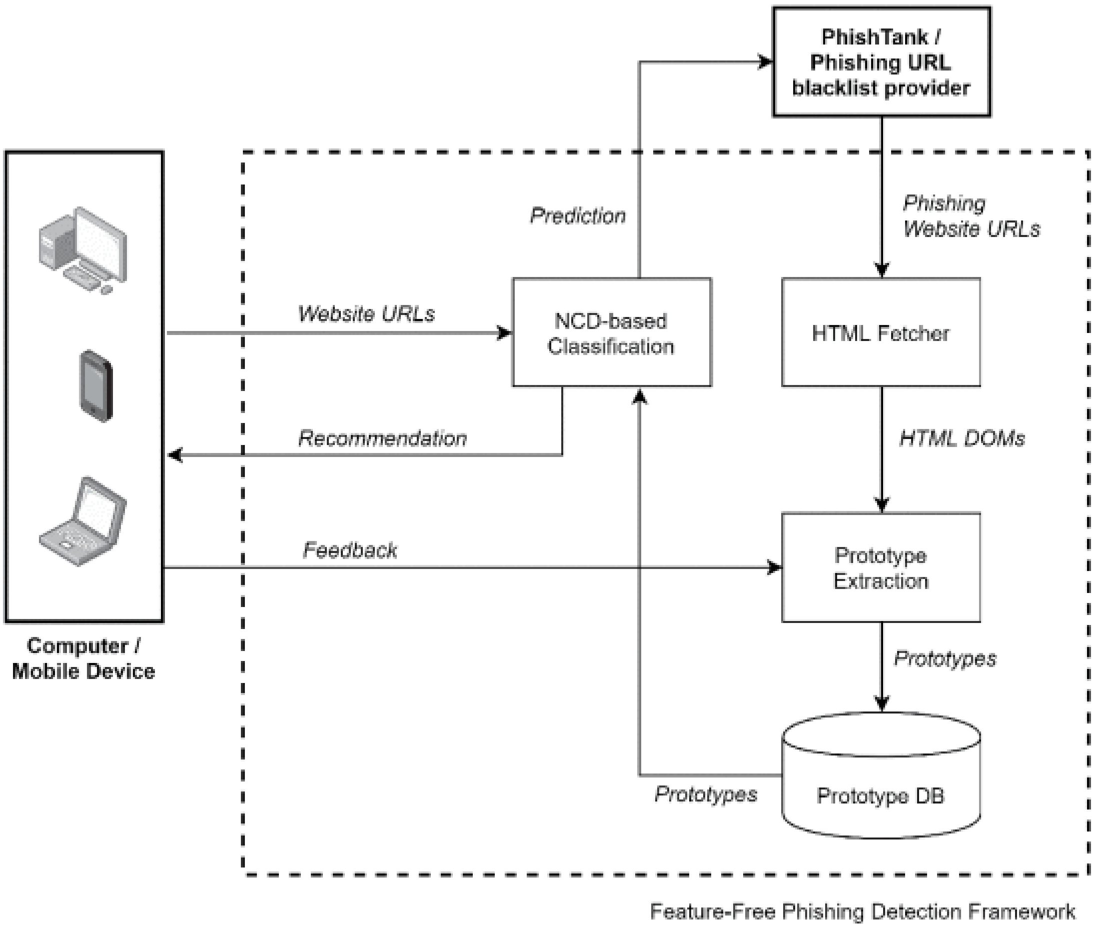

Figure 2: System Diagram of the Feature-free Detection

C. A Deep Learning – Based Framework

A Deep-Learning-Based Framework was suggested by Lizhen's research[3]. The study proposes the following methods: Random Forest, Logistic Regression, SVM, Recurrent Neural Network

(RNN), RNN-LSTM (RNN-Long Short Term Memory), and RNN-GRU (RNN-Gated Recurrent Unit). The research method, which consists of six steps, is depicted in Figure 3. They began by compiling information from a variety of sources, including Phish Storm, Phish Tank, ISCX 2016, Kaggle, and others. Second, for model training, various datasets will be combined. Third, create an API for predicting the risk of phishing. Fourth, the prediction interface is called by the browser extension for real-time detection, and the results are shown. Fifth, when users disagree with detection results, such as a wrong guess or missed alarm, they can provide real-time feedback. After validation, the user's report is manually and automatically examined, and the results are synchronized with the data set. The accuracy and performance of the proposed models are as follows: SVM had 98.85%, Logistic Regression had 98.89%, Random Forest had 98.5%, Random Forest-GRU had 99.18%, and Random Forest-LSTM had 98.95%.

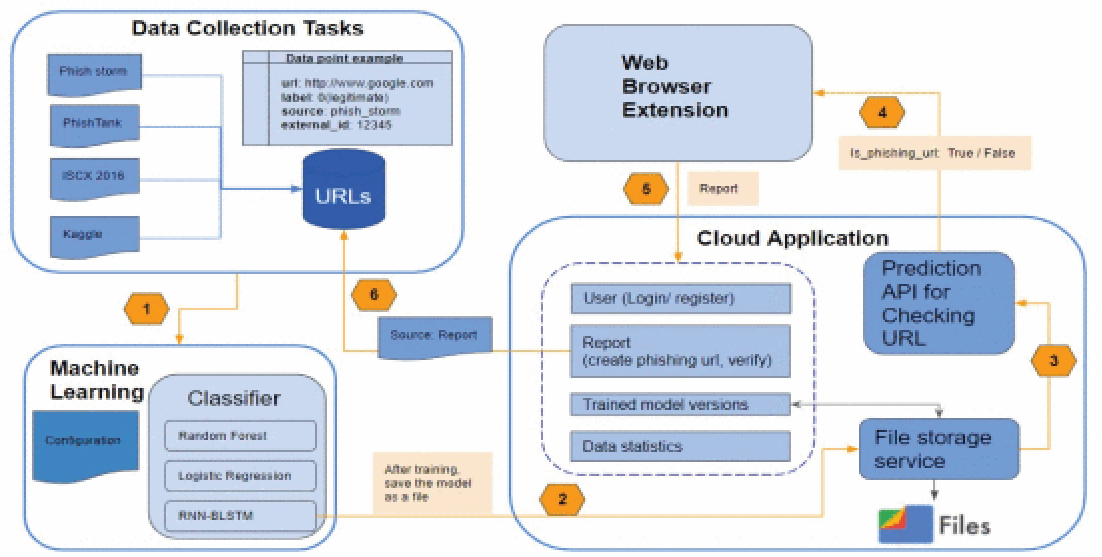

Figure 3: Architecture of the deep learning-based framework

D. Multilayer Stacked Ensemble Learning Model

An Ensemble Learning Method to Detect Phishing was developed in the study by Lakshmana Rao [4]. The MLSELM(Multi-Layer Stacked Ensemble Learning Model) is the research method provided. The three layers of this model are shown in Figure 4. Classification techniques like XG Boost (XGB), Random Forests(RF), Logistic Regression, MLP(Multi-Layer Perceptron), and KNN are included in the First Layer. RF, XGB, and MLP make up the second layer, and XGB, which is the Meta Layer, makes up the final layer. Four datasets, obtained from Mendeley and the UCI repository, are included in the proposed effort. There are three parts to the planned work. In the first place, is the Information Stage, in which the phishing dataset is utilized as info. The Data Balancing Phase, which transforms imbalanced data into balanced data, comes in second. Finally, the implementation of the model. The performance and accuracy of the model across datasets are as follows: Dataset 1: 97.76 percent, Dataset 2: Dataset 3: 98.90% 95.69 %, and Dataset 4: 98.43%.

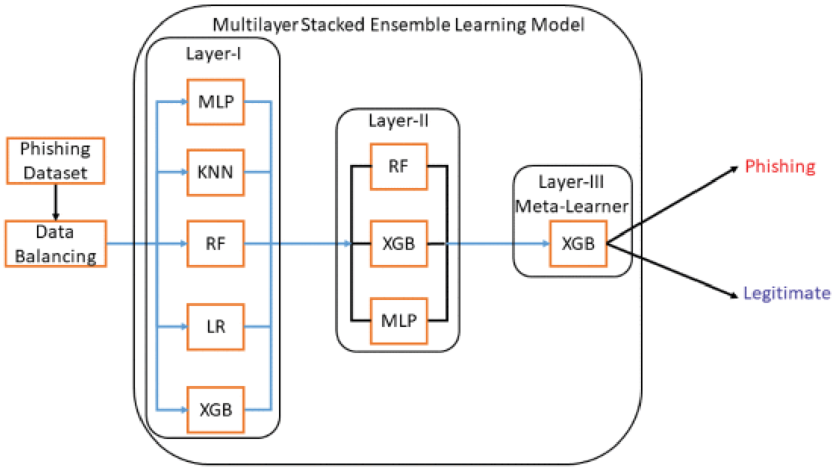

Figure 4: Multi-Layer Stacked Ensemble Learning Model

E. Multidimensional Features driven by Deep Learning

Peng[5] suggested a phishing detection technique based on deep learning in their research. The Multidimensional Feature Phishing Detection Model (MFPD) is the suggestion made in the study. It is based on a quick deep-learning detection method. This model is made up of three algorithms, as shown in Figure 5. DCDA(Dynamic Category Decision Algorithm), CNN-LSTM, and MFA (Multidimensional Feature Algorithm). Phish Tank and DMOZ Tools were used to compile the actual dataset in the Proposed Work. Phishing is detected using a two-category processing method. The URL is first entered into the CNN-LSTM Layer, and then the Layer's output is given to the DCDA Algorithm as an input. The DCDA Algorithm checks for specific requirements to decide whether or not to move on to MFA. The performance and accuracy of the proposed method are 99.41%.

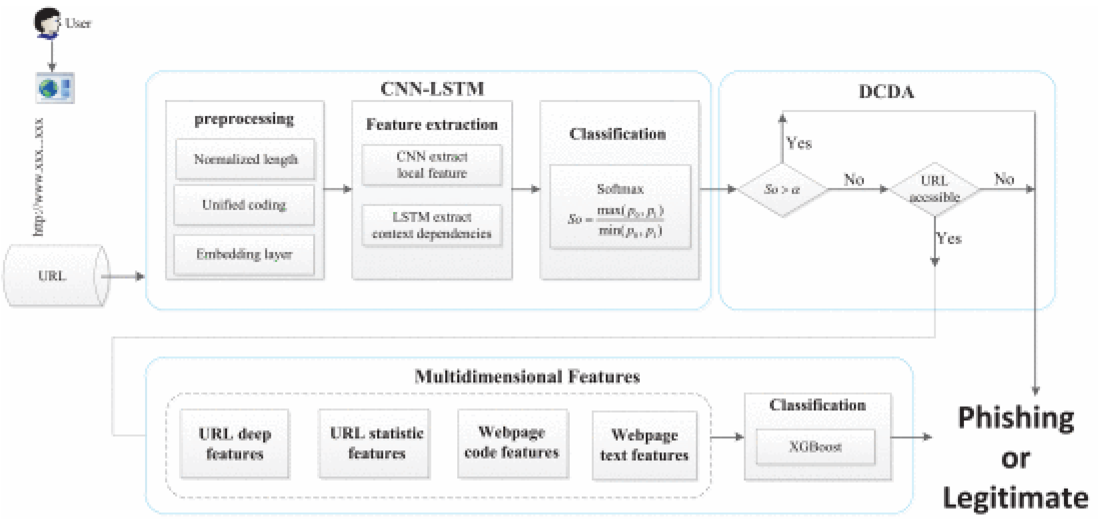

Figure 5: Multidimensional Features driven by Deep Learning Framework

1.2 ORGANIZATION OF THE REPORT

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, and software details.
- Chapter 3 discusses the results obtained after the project was implemented.
- Chapter 4 concludes the report.
- Chapter 5 consists of codes.
- Chapter 6 gives references.

CHAPTER – 2

WEBSITE PHISHING DETECTION

This chapter describes the working methodology, software, and hardware details.

2.1 Working Methodology

This section covers both the general research approach and the experimental framework. The proposed phishing detection models of Logistic Regression, DT, SVM, Random Forests, KNN, Neural Networks, Naïve Bayes, and Genetic Algorithms are then discussed, as are the dataset properties.

2.1.1 Overall Research Methodology

This study approaches the problem of website phishing detection as an AI-based binary-classification problem, with the decision-making step result identifying whether a particular website is a real or fake website. Consider using ML Algorithms as the foundation for developing credible and practical phishing detection models to combat phishing attacks. These proposed ML Algorithms will address the issues raised Logistic Regression, DT, SVM, Random Forests, KNN, Neural Networks, Naïve Bayes, and Genetic Algorithms were the AI Algorithms chosen for this project.

The overall strategy is broken down into three stages. The first step is to collect and analyze the experimental data. The website phishing dataset from this study is widely used in other studies. This same dataset can be found on Kaggle and UCI Datasets. There are 11,055 occurrences in this dataset, 30 independent features, and 1 class variable with two labels i.e., {"1": "Legitimate Website", "-1": "Phishing Website"}.

The 30 independent variables are divided into four categories:

- Address Bar Features: It has 12 features out of 30.
- Abnormal Features: It has 6 out of 30 features.
- JavaScript and HTML-based Features: It has 5 out of 30 features.
- Domain-based Features: It has 7 features out of 30.

2.1.2 Proposed Phishing Detection Models

In this research, the proposed website phishing detection methods are referred to as Logistic Regression, DT, SVM, Random Forests, KNN, Neural Networks, Naïve Bayes, and Genetic Algorithms.

The LR model[11] computes the logistic of the output by adding the input characteristics. The LR model's output is always between 0 and 1. The goal of DT[12] is to create a trained model that can anticipate the class or value of the target class by learning basic decision rules from previous data. RF[13] is a Bagging method approach that involves randomly picking subsets of training data, fitting a model in a smaller dataset, and aggregating the results. SVM[14] seeks the optimal line between two dimensions or the best hyperplane that splits space into classes. KNN[15] is used to predict test data based on the features of the data points. This is accomplished by computing the distance between the test and training data, assuming that comparable objects exist nearby. NB[16] is a supervised machine learning method influenced by the Bayes theorem. It operates on the conditional probability principle. To produce a prediction, NB uses the probability of each attribute belonging to each class. NN[17] is a computational algorithm used for dataset categorization. A GA[18] uses a binary string representation to characterize prospective issue hypotheses and iterates through a search space of viable hypotheses in an attempt to find the "best hypothesis," which is the one that maximizes a preset numerical measure, or fitness. GAs are a subset of evolutionary algorithms as a whole.

2.1.3 Algorithms

- **Logistic Regression**

Forward Pass Equation:

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

Where X is the input data and θ is the parameter, we want to learn or train or optimize.

Optimization / Loss Equation:

$$J(\theta) = -\frac{1}{m}\sum_{m}^{i=1}\left(y^i \, log\left(p^i\right) + \left(1 - y^i\right) log\left(1 - p^i\right)\right)$$

Where p^i is the prediction value of the ith sample, m is the number of samples in the training data, and y^i is the label of the ith sample.

End

- **Decision Tree:**
    - The root node of the tree is the real dataset S at the start.
    - For each iteration, the approach calculates the IG(Information Gain) and H(Entropy) of the extremely underutilized attribute of the set S.
    - The attribute with the highest "IG" or the lowest "H" is then chosen.
    - To create a subset of the data, the set S is divided by the chosen attribute.
    - As the algorithm iterates over each subset, only traits that have never been chosen before are considered.

Formula for Entropy(H) is:

$$E(S) = \sum_{i=1}^{c} -p_i \, log_2 \, p_i$$

The formula for Information Gain

$$Information\ Gain = Entropy(before) - \sum_{j=1}^{K} Entropy(j, after)$$

- **Random Forest:**

Random Forests are created in the following steps:

Step 1: At random, choose "k" features from a total of "m" features, where k << m

Step 2: Determine the node "d" by using the best split point among the "k" features.

Step 3: The best split creates daughter nodes from the parent node.

Step 4: Repeat steps 1-3 until you reach the "l" number of nodes.

Step 5: Make a forest by repeating steps 1–4 "n" times to make "n" trees.

- **Support Vector Machine**

Forward Pass Equation:

For **RBF Kernel** the equation:

$$K(x,x') = e^{-\gamma\|x-x'\|^2}$$

$$\gamma = \frac{1}{nfeatures \times \sigma^2}$$

Where,

$\|x - x'\|^2$ is the distance(Euclidean) squared between two data vectors (2 points) and γ is an influencing factor for a single training example (point).

For **Sigmoid Kernel** the equation:

$$K(x_i, x_j) = \tanh(a x_i^T x_j + r)$$

Where x_i and x_j are two feature vectors

- **K- Nearest Neighbour Algorithm:**

Forward Pass Equation:

For Euclidean Distance:

$$D(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Where x and y are two data points

For Manhattan Distance:

$$D(x,y) = \sum_{i=1}^{k}|x_i - y_i|$$

Where x and y are two data points

- **Naïve Bayes Algorithm:**

Bayes Theorem:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

For Bernoulli NB[19]:

$$P(x_i|y) = P(i|y)x_i + (1 - p(i|y))(1 - x_i)$$

- **Neural Network:**

For Layers 1 and 2, the activation is **ReLU**, and the equation of the function is:

$$y = max(0, x)$$

For Final Layer the activation is **Sigmoid**, and the equation of the function is:

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

The loss function is the **Binary CrossEntropy function**, and the equation of the loss is:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \,.\, log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

And the Optimization function is **Adam**, the equations for Adam are:

For Each Parameter ω_j

$$v_t = \beta_1 \times v_{t-1} - (1 - \beta_1) \times g_t$$

$$s_t = \beta_2 \times s_{t-1} - (1 - \beta_2) \times g_t^2$$

$$\Delta\omega_t = -\eta\left(\frac{v_t}{\sqrt{s_t + \epsilon}} \times g_t\right)$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η: Initial Learning Rate
g_t: Gradient at time t along ω_j
v_t: Exponential Average of gradients along ω_j
s_t: Exponential Average of squares of gradients along ω_j
β_1, β_2: Hyperparameters

- **Genetic Algorithm:**

The Algorithm is as follows:

- Generation of Initial Population
- Calculation of Fitness of Individuals
- Checking of the Stop Criteria

- If Yes End the Algorithm
- Else Selection of the individuals

▪ If Else then Selection of the Genetic Operator

- Crossover Operator: Swap a gene between the individuals
- Mutation Operator: Mutate the genes in an individual

▪ Generate the population until the Stop Criteria is satisfied

The flowchart of the above algorithm is in figure 6.

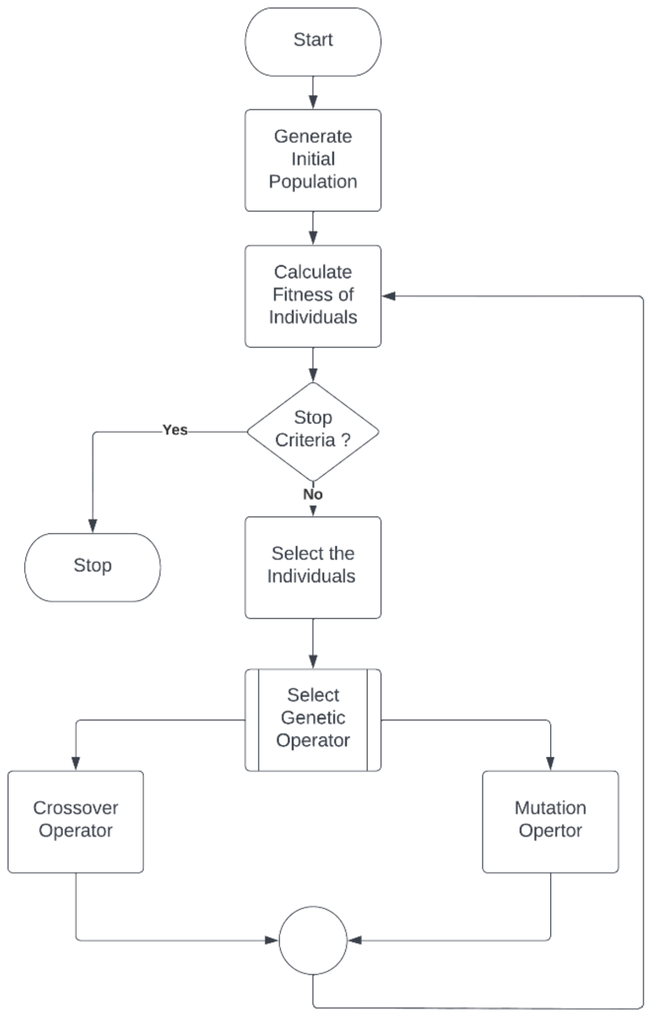

Figure 6 : Flowchart of NEAT Algorithm

2.2 SYSTEM DETAILS

2.2.1 SOFTWARE DETAILS

- Python 3.10.6 or above, PIP, NumPy, Pandas, Jupyter, TensorFlow, Keras.

Python:

The general-purpose, interactive, object-oriented, and high-level programming language Python is particularly well-liked. It is a garbage-collected and dynamically typed programming language. Between 1985 and 1990, Guido van Rossum designed it. Python source code is also accessible under the GNU General Public License, just like Perl (GPL).

Programming languages such as procedural, object-oriented, and functional are supported by Python. Python's design philosophy places a strong emphasis on code readability

- Python is straightforward to learn. Python is adaptable and can be used to construct a wide variety of things.
- Python includes strong development libraries, such as those for AI and ML.
- Python is in high demand and pays well

PIP:

Pip is a widely used package manager for Python that is used to install and maintain packages. There is a selection of built-in functions and built-in packages included with the Python standard library. The Python standard library does not include data science libraries like scikit-learn and statsmodel. They can be set up using the command line and pip, the default package manager for Python.

NumPy:

The Python package NumPy is used to manipulate arrays. Additionally, it provides functions for working with matrices, the Fourier transform, and the linear algebra domain. In the year 2005, Travis Oliphant developed NumPy. You can use it for free because it is an open-source project. Numerical Python is referred to as NumPy.

- Lists can function like arrays in Python, but they take a long time to execute.
- NumPy seeks to offer array objects that are up to 50 times faster than Python lists.
- The NumPy array object is referred to as ndarray, and it has several accompanying methods that make using ndarray very simple.

- Data science, where efficiency and availability of resources are crucial, commonly uses arrays.

Pandas:

The most widely used Python data analysis library is called Pandas. With back-end source code that is solely written in C or Python, it offers highly optimized performance. Data analysis in Pandas is possible with Series and DataFrames.

TensorFlow:

The software library TensorFlow is free and open-source, and it may be used for different programming and dataflow across a variety of tasks. It is a library for symbolic math, and neural networks and other machine learning techniques use it.

Keras:

An open-source neural network library created in Python is called Keras. It can be used with TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or Plaid ML as a foundation. Its user-friendliness, modularity, and extensibility are its main design goals as it aims to facilitate quick experimentation with deep neural networks. As well as a variety of tools to make working with image and text data easier, Keras includes numerous implementations of widely used neural network building blocks like layers, objectives, activation functions, and optimizers. This helps to simplify the coding required to create deep neural networks.

2.2.2 HARDWARE CONFIGURATION

- **Processor:** Intel Core i5 – 6200U, Dual-Core, 2.3GHz minimum speed per core
- **RAM:** 8GB
- **Hard Disk:** 10GB or more available space
- **Operating System:** Windows 10 Pro

CHAPTER 3

RESULTS AND DISCUSSIONS

ACCURACY / PERFORMANCE OF THE MODELS

The website phishing dataset is partitioned into 80:20 training and testing sets. Each model is trained using a training set, and the testing set is used to evaluate model performance. The model's performance was assessed by computing the model's AS(accuracy score), FNR (false negative rate), and FPR(false positive rate).

Table 1: Performance of Proposed Algorithms

Classification	Accuracy Score	False Positive Rate	False Negative Rate
Logistic Regression	92.40%	0.096	0.059
KNN using Euclidean Distance	94.68%	0.066	0.042
KNN using Manhattan Distance	95.36%	0.063	0.032
SVM using RBF Kernel	94.35%	0.082	0.035
SVM using Sigmoid Kernel	83.17%	0.212	0.133
Decision Tree Classifier	95.69%	0.051	0.034
Random Forest Classifier	97.03%	0.043	0.018
Naïve Bayes Algorithm	90.66%	0.095	0.091
Neural Network	97.43%	0.028	0.023
Genetic Algorithm	85.53%	0.311	0.057

LOGISTIC REGRESSION

```
logr = CR(y_test,y_pr)
print(logr)
cm = CM(y_test,y_pr)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
```

```
              precision    recall  f1-score   support

          -1       0.92      0.90      0.91      1225
           1       0.92      0.94      0.93      1539

    accuracy                           0.92      2764
   macro avg       0.92      0.92      0.92      2764
weighted avg       0.92      0.92      0.92      2764

9.63265306122449 5.977907732293697
```

K-NN using Euclidean Distance

```
y_pr1 = kn1.predict(X_test)
kn1_eu = CR(y_test,y_pr1)
print(kn1_eu)
cm = CM(y_test,y_pr1)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr1)
```

```
              precision    recall  f1-score   support

          -1       0.95      0.93      0.94      1225
           1       0.95      0.96      0.95      1539

    accuracy                           0.95      2764
   macro avg       0.95      0.95      0.95      2764
weighted avg       0.95      0.95      0.95      2764

6.693877551020408 4.223521767381416

0.9468162083936325
```

K-NN using Manhattan Distance

```
kn2 = KNC(n_neighbors=5,metric='minkowski',p=1)
kn2.fit(X_train,y_train)
y_pr2 = kn2.predict(X_test)
kn2_eu = CR(y_test,y_pr1)
print(kn2_eu)
cm = CM(y_test,y_pr2)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr2)
```

```
              precision    recall  f1-score   support

          -1       0.95      0.93      0.94      1225
           1       0.95      0.96      0.95      1539

    accuracy                           0.95      2764
   macro avg       0.95      0.95      0.95      2764
weighted avg       0.95      0.95      0.95      2764

6.36734693877551 3.248862897985705

0.9536903039073806
```

SVM using RBF Kernel

```
from sklearn.svm import SVC
sm1 = SVC(kernel='rbf')
sm1.fit(X_train,y_train)
y_pr3 = sm1.predict(X_test)
sm1_rbf = CR(y_test,y_pr3)
print(sm1_rbf)
cm = CM(y_test,y_pr3)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr3)
```

```
              precision    recall  f1-score   support

          -1       0.95      0.92      0.94      1225
           1       0.94      0.96      0.95      1539

    accuracy                           0.94      2764
   macro avg       0.94      0.94      0.94      2764
weighted avg       0.94      0.94      0.94      2764

8.244897959183675 3.5737491877842755

0.9435600578871202
```

SVM using Sigmoid Kernel

```
from sklearn.svm import SVC
sm2 = SVC(kernel='sigmoid')
sm2.fit(X_train,y_train)
y_pr4 = sm2.predict(X_test)
sm2_sig = CR(y_test,y_pr4)
print(sm2_sig)
cm = CM(y_test,y_pr4)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr4)
```

```
              precision    recall  f1-score   support

          -1       0.82      0.79      0.81      1225
           1       0.84      0.87      0.85      1539

    accuracy                           0.83      2764
   macro avg       0.83      0.83      0.83      2764
weighted avg       0.83      0.83      0.83      2764

21.224489795918366 13.32033788174139

0.8317655571635311
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier as DTC
dtc = DTC()
dtc.fit(X_train,y_train)
y_pr5 = dtc.predict(X_test)
dec_tr = CR(y_test,y_pr5)
print(dec_tr)
cm = CM(y_test,y_pr5)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr5)
```

```
              precision    recall  f1-score   support

          -1       0.96      0.95      0.95      1225
           1       0.96      0.96      0.96      1539

    accuracy                           0.96      2764
   macro avg       0.96      0.96      0.96      2764
weighted avg       0.96      0.96      0.96      2764

5.3061224489795915 3.508771929824561

0.9569464544138929
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier as RFC
rfc = RFC(n_estimators=10,random_state=0,n_jobs=1)
rfc.fit(X_train,y_train)
y_pr6 = rfc.predict(X_test)
ran_for = CR(y_test,y_pr6)
cm = CM(y_test,y_pr6)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr6)
```

```
4.326530612244897 1.8843404808317088

0.9703328509406657
```

Naïve Bayes Algorithm

```
from sklearn.naive_bayes import BernoulliNB as BN
bn = BN()
bn.fit(X_train,y_train)
y_pr7 = bn.predict(X_test)
ber_nb = CR(y_test,y_pr7)
print(ber_nb)
cm = CM(y_test,y_pr7)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr7)
```

```
              precision    recall  f1-score   support

          -1       0.89      0.90      0.90      1225
           1       0.92      0.91      0.92      1539

    accuracy                           0.91      2764
   macro avg       0.90      0.91      0.91      2764
weighted avg       0.91      0.91      0.91      2764

9.551020408163264 9.161793372319687

0.9066570188133141
```

Neural Network

```
cm = CM(y_test,predictions)
print(cm)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,predictions)
```

```
[[1190   35]
 [  36 1503]]
2.857142857142857 2.3391812865497075

0.9743125904486252
```

Genetic Algorithm

```
weights =  ag.get_best_individua()
nn = NeuralNetwork(sizes=[30, 10, 5, 1], weights = weights)
print(nn.prediction_score(x_test,y_test))
```

```
0.8553345388788427
```

From the table and figures, we can see that Neural Network gives better accuracy than other algorithms i.e., 97.43% and the lowest False Positive Rate i.e., 0.028, but Random Forest Classifier has the lowest False Negative Rate i.e., 0.018.

CHAPTER 4

CONCLUSION

The goal of this research is to provide an ideal solution to today's phishing problem. This study aims to address any remaining flaws that have been identified in this manner. Simple methods' inability to detect phishing methods accurately, high FPR and FNR, ensemble methods' inability to detect phishing websites excellently, and the poor performance of some hybridized methods in detecting phishing websites when compared to simple classification models are among the shortcomings previously identified in Section II. The need to address these issues drove the pursuit of this research project. This study yielded eight (8) different ML, DL, and GA algorithms with high AS and low FPR. One of the approaches suggested has an extremely high AS of around 97.4%, as well as a low FPR of 0.028 (Neural Network) and a low FNR of 0.018. (Random Forest Algorithm). The findings show that the proposed approaches are useful and efficient, with a low rate of false alarms while achieving high AS and F-measure scores.

CHAPTER 5

APPENDIX

CODES

All Proposed Models excluding Genetic Algorithm:

```
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
from sklearn import preprocessing
import keras
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import accuracy_score as AS
from sklearn.metrics import classification_report as CR
from sklearn.metrics import confusion_matrix as CM
from os import path

"""## Step 1: Data Preprocessing"""

df = pd.read_csv('Phishing-Websites-Detection-master\Phishing.csv')
df.head()

"""#### ***Null Value Checker***"""

print(df.info())
print("\n")
df.isnull().any()

"""#### *Hence there are no Null Values in the dataset*


#### ***Checking for Dependent and Independent Variables***
"""

X = df.copy()

y = X.pop('Result')

X = df.drop(columns=['Result'])

y

"""#### *Here the Last Column named as Result in Dataset is the **dependent
variable** which is dependent of 30 no of Factors*
#### *Other than Result column are **independent variable***

### ***Splitting the dataset into Train and Test sets***
```

```
"""

X_train,X_test,y_train,y_test = tts(X,y,stratify=y,test_size=0.25)
input_shape = [X_train.shape[1]]
print("Input shape is ", input_shape)
X.head()

"""### ***Logistic Regression***"""

from sklearn.linear_model import LogisticRegression as LR
lr = LR()
lr.fit(X_train,y_train)

y_pr = lr.predict(X_test)

logr = CR(y_test,y_pr)
print(logr)
cm = CM(y_test,y_pr)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)

"""### ***K- Nearest Neighbour Algorithm using Euclidean Distance***"""

from sklearn.neighbors import KNeighborsClassifier as KNC
kn1 = KNC(n_neighbors=3,metric='minkowski',p=2)
kn1.fit(X_train,y_train)

y_pr1 = kn1.predict(X_test)
kn1_eu = CR(y_test,y_pr1)
print(kn1_eu)
cm = CM(y_test,y_pr1)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr1)

"""### ***K-Nearest Neighbour Algorithm using Manhattan Distance***"""

kn2 = KNC(n_neighbors=5,metric='minkowski',p=1)
kn2.fit(X_train,y_train)
y_pr2 = kn2.predict(X_test)
kn2_eu = CR(y_test,y_pr1)
print(kn2_eu)
cm = CM(y_test,y_pr2)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr2)
```

```
"""### ***Support Vector Machine using RBF Kernel***"""

from sklearn.svm import SVC
sm1 = SVC(kernel='rbf')
sm1.fit(X_train,y_train)
y_pr3 = sm1.predict(X_test)
sm1_rbf = CR(y_test,y_pr3)
print(sm1_rbf)
cm = CM(y_test,y_pr3)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr3)

"""### ***Support Vector Machine using Sigmoid Kernel***"""

from sklearn.svm import SVC
sm2 = SVC(kernel='sigmoid')
sm2.fit(X_train,y_train)
y_pr4 = sm2.predict(X_test)
sm2_sig = CR(y_test,y_pr4)
print(sm2_sig)
cm = CM(y_test,y_pr4)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr4)

"""### ***Decision Tree Classifier***"""

from sklearn.tree import DecisionTreeClassifier as DTC
dtc = DTC()
dtc.fit(X_train,y_train)
y_pr5 = dtc.predict(X_test)
dec_tr = CR(y_test,y_pr5)
print(dec_tr)
cm = CM(y_test,y_pr5)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr5)

"""### ***Random Forest Classifier***"""

from sklearn.ensemble import RandomForestClassifier as RFC
rfc = RFC(n_estimators=10,random_state=0,n_jobs=1)
rfc.fit(X_train,y_train)
y_pr6 = rfc.predict(X_test)
```

```
ran_for = CR(y_test,y_pr6)
cm = CM(y_test,y_pr6)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr6)

"""### ***Naïve Bayes Algorithm***"""

from sklearn.naive_bayes import BernoulliNB as BN
bn = BN()
bn.fit(X_train,y_train)
y_pr7 = bn.predict(X_test)
ber_nb = CR(y_test,y_pr7)
print(ber_nb)
cm = CM(y_test,y_pr7)
# CP(y_test,y_pr)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,y_pr7)

"""### ***Neural Network***"""

from keras.models import Sequential
from keras.layers import Dense
from keras import layers
import keras



df['Result'] = df['Result'].map({-1:0, 1:1})
df['Result'].unique()

X = df.copy()

y = X.pop('Result')

X = df.drop(columns=['Result'])

y
X_train,X_test,y_train,y_test = tts(X,y,stratify=y,test_size=0.25)
input_shape = [X_train.shape[1]]
print("Input shape is ", input_shape)
X.head()

model = Sequential([
    layers.BatchNormalization(input_shape=input_shape),
    Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
```

```python
    Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    Dense(1, activation='sigmoid'),
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy',keras.metrics.Precision(),keras.metrics.Recall()],

)

early_stopping = keras.callbacks.EarlyStopping(
    patience=20,
    min_delta=0.01,
    restore_best_weights=True,
)
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    batch_size=64,
    epochs=100,
    callbacks=[early_stopping],
)

history_df = pd.DataFrame(history.history)
history_df.loc[0:, ['loss', 'val_loss']].plot()
history_df.loc[0:, ['binary_accuracy', 'val_binary_accuracy']].plot()
print(history_df)
print(f"Best Validation Loss: {history_df['val_loss'].min()}" +\
      f"\nBest Validation Accuracy: {history_df['val_binary_accuracy'].max()}"+\
      f"\nBest Recall: {history_df['val_recall'].max()}" +\
      f"\nBest Precision: {history_df['val_precision'].max()}"
)

predictions = model.predict(X_test)
predictions

predictions = (predictions > 0.5)*1

cm = CM(y_test,predictions)
print(cm)
FPR = (cm[0][1]/(cm[0][0] + cm[0][1]))*100
FFR = (cm[1][0]/(cm[1][0]+cm[1][1]))*100
print(FPR,FFR)
AS(y_test,predictions)
```

Genetic Algorithm:

```
# -*- coding: utf-8 -*-
# Commented out IPython magic to ensure Python compatibility.
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.metrics import accuracy_score
import random
import operator
import copy


# %matplotlib inline
import warnings
warnings.filterwarnings('ignore')

df1= pd.read_csv('Phishing-Websites-Detection-master\Phishing.csv')
df1



# df1 = df.head(500)
#

# df=pd.get_dummies(df1)
df1
# df1.rename(columns={'Result': 'Class'}, inplace=True)

df1['Result'] = df1['Result'].map({-1:0, 1:1})
df1['Result'].unique()





class NeuralNetwork():
  def __init__(self, sizes, weights):
    self.sizes = sizes
    self.params = weights

  def sigmoid(self, x, derivative=False):
    if derivative:
      return (np.exp(-x))/((np.exp(-x)+1)**2)
    return 1/(1 + np.exp(-x))

  def softmax(self, x, derivative=False):
    # Numerically stable with large exponentials
```

```
    exps = np.exp(x - x.max())
    if derivative:
      return exps / np.sum(exps, axis=0) * (1 - exps / np.sum(exps, axis=0))
    return exps / np.sum(exps, axis=0)

  def forward_pass(self, x_train):
    params = self.params

    # input layer activations becomes sample
    params['A0'] = x_train

    # input layer to hidden layer 1
    params['Z1'] = np.dot(params["W1"], params['A0'])
    params['A1'] = self.sigmoid(params['Z1'])

    # hidden layer 1 to hidden layer 2
    params['Z2'] = np.dot(params["W2"], params['A1'])
    params['A2'] = self.sigmoid(params['Z2'])

    # hidden layer 2 to output layer
    params['Z3'] = np.dot(params["W3"], params['A2'])
    params['A3'] = self.sigmoid(params['Z3'])

    return params['A3']



  def compute_accuracy(self, x_val, y_val):
    '''
        This function does a forward pass of x, then checks if the indices
        of the maximum value in the output equals the indices in the label
        y. Then it sums over each prediction and calculates the accuracy.
    '''
    predictions = []
    for x, y in zip(x_val, y_val):
      output = self.forward_pass(x)
      pred = np.argmax(output)
      predictions.append(pred == np.argmax(y))

    return np.mean(predictions)

  def prediction_value(self, x_val, y_val):
    '''
        This function does a forward pass of x, then checks if the indices
        of the maximum value in the output equals the indices in the label
        y. Then it sums over each prediction and calculates the accuracy.
    '''
    predictions = []
    for x, y in zip(x_val, y_val):
      output = self.forward_pass(x)
      pred = np.argmax(output)
      predictions.append(pred == np.argmax(y))
```

```
    return predictions


  def prediction_score(self, x_val, y_val):
    '''
        This function does a forward pass of x, then checks if the indices
        of the maximum value in the output equals the indices in the label
        y. Then it sums over each prediction and calculates the accuracy.
    '''
    outputs = []
    for x, y in zip(x_val, y_val):
      output = self.forward_pass(x)
      #pred = np.argmax(output)
      pred = np.where(output[0] > 0.5, 1, 0)


      outputs.append(pred)
      #predictions.append(pred == np.argmax(y))

    accuracy = accuracy_score(y_val, outputs)
    return accuracy

class AG(object):
  def __init__(self, instance):
    self.instance = instance
    self.fitness_stat = []
    self.generation_stat = []
    self.time_stat = []
    self.mean_stat = []
    self.counter_time = 0
    self.counter = 0


  '''Executing the algorithm'''
  def run(self):

    ''' the initial population '''
    population = self.instance.initializePopulation()


    ''' General cycle of the genetic algorithm '''
    while 1:
      start = time.process_time()

      ''' Evaluation of the individuals of the population'''

      self.counter+=1
      fitness_population = []
      mean_fitness= 0
      for individual in population:
```

```
      fitness = self.instance.getFitness(individual)
      val = {'fitness':fitness , 'individual' : individual}
      mean_fitness += fitness
      fitness_population.append(val)

    self.mean_stat.append(mean_fitness/len(population))
    print("Generation", self.counter, "Accuracy",mean_fitness/len(population))

    '''record the statistics'''
    self.fitness_stat.append(self.instance.getFitnessStat())
    self.generation_stat.append(self.instance.getGenerationStat())
    self.time_stat.append(self.counter_time)


    ''' Best solution and check the exit criteria'''
    if self.instance.showAndCheck(fitness_population):
      break

    ''' Advance to the next town'''
    population = self.changePopulation(fitness_population);

    interval = time.process_time() - start
    self.counter_time  = self.counter_time + interval



''' Change from old population to the new one'''
def changePopulation(self,fitness_population):
  ''' Selection of the best individuals (using the tournament method) '''
  parentsGenerator = self.instance.selectParents(fitness_population)


  allChildren = []
  ''' create new population '''
  while len(allChildren) < len(fitness_population):
    parents = next(parentsGenerator) # Next yielded pair of parents

    '''Probability for Crossing'''
    if random.random() > self.instance.getCrossThreshold():
      children = self.instance.crossover(parents)
    else:
      children = parents
    '''Probability for the mutation'''
    for child in children:
      if random.random() > self.instance.getMutationThreshold():
        allChildren.append(self.instance.mutate(child))
      else:
        allChildren.append(child)

  '''New population'''
  return allChildren[:len(fitness_population)] # May exceed
```

```
  ''' print_result '''
  def get_best_individua(self):
    return self.instance.bestIndividual

  def print_best_result(self):
    print(f"Generation {self.instance.bestGeneration} Best Solution:
{self.instance.bestIndividual} with fitness: {self.instance.bestFitness} ")

  ''' graban las estadisticas'''
  def result_statistics(self):
    plt.figure(figsize=(6,5))
    plt.plot(self.generation_stat, self.fitness_stat,'bo-')
    plt.xlabel('Generation ', fontsize=14)
    plt.ylabel('Best Solution', fontsize=14)
    plt.title( f"Mutation Rate.{self.instance.getMutationThreshold()}
%,  Crossing Rate. {self.instance.getCrossThreshold()}%, Last generation
{self.generation_stat[-1]}", fontsize=14)
    plt.show()

  def time_statistics(self):
    plt.figure(figsize=(6,5))
    plt.plot(self.generation_stat, self.time_stat,'bo-')
    plt.xlabel('Generation ', fontsize=14)
    plt.ylabel('Time ', fontsize=14)
    plt.title( " Time ", fontsize=14)
    plt.show()

  def mean_statistics(self):
    plt.figure(figsize=(6,5))
    plt.plot(self.generation_stat, self.mean_stat,'bo-')
    plt.xlabel('Generation ', fontsize=14)
    plt.ylabel('Accuracy ', fontsize=14)
    plt.title( " Accuracy  ", fontsize=14)
    plt.show()

class rules(object):

  ''' params '''

  def __init__(self, maxLoopsNum, error, populationSize, crossThreshold,
mutationThreshold,X,y,structure):
    self.counter = 0
    self.maxLoopsNum = maxLoopsNum
    self.error = error
    self.populationSize = populationSize
    self.crossThreshold = crossThreshold
    self.mutationThreshold = mutationThreshold
    self.fitness_stat = 0
    self.generation_stat = 0
    self.bestIndividual = list()
    self.bestFitness = 0
    self.bestGeneration = 0
```

```
    self.structure = structure
    self.X = X
    self.y = y



  def getMutationThreshold(self):
    return self.mutationThreshold

  def getCrossThreshold(self):
    return self.crossThreshold

  def getMaxLoopsNum(self):
    return self.maxLoopsNum

  def getFitnessStat(self):
    return self.fitness_stat

  def getGenerationStat(self):
    return self.generation_stat

  def getLastIndividual(self):
    return self.bestIndividual

  ''' Initial population '''
  def initializePopulation(self):
    population = [];
    for i in range(self.populationSize):
      population.append(self.generate_weights_individu(self.structure))
    return population

  def getFitness(self, individual):
    nn =   NeuralNetwork(sizes=self.structure, weights = individual)
    score = nn.prediction_score(self.X, self.y)
    return  score

  def showAndCheck(self, fitness_population):
    self.counter += 1
    fitness_population.sort(key=operator.itemgetter('fitness'))
    # Printing Generation Fitness
    #print("Generation", self.counter, "Best solution:", fitness_population[-
1]['fitness'])

    # Saving The Stats
    self.fitness_stat = fitness_population[-1]['fitness']
    self.generation_stat = self.counter
    best = fitness_population[-1]  # best Individual


    # best solution
    if (fitness_population[-1]['fitness'] >= self.bestFitness):
      self.bestIndividual = fitness_population[-1]['individual']
```

```
      self.bestFitness = fitness_population[-1]['fitness']
      self.bestGeneration = self.counter

    check = (self.counter > self.maxLoopsNum) or (fitness_population[-
1]['fitness']>= self.error)
    return check

  def selectParents(self, fitness_population):
    # Construct a iterator here
    # Use Tournament Selection
    while 1:
      parent1 = self.tournament(fitness_population)
      parent2 = self.tournament(fitness_population)
      yield (parent1, parent2)

  def crossover(self, parents):
    parent1, parent2 = parents
    father_1, father_2 = parents
    chlid_1 = {
      'W1': father_1['W1'],
      'W2': father_2['W2'] ,
      'W3': father_2['W3']
    }

    chlid_2 = {
      'W1': father_2['W1'],
      'W2': father_2['W2'] ,
      'W3': father_1['W3']
    }
    return (chlid_1,  chlid_2)

  def mutate(self, child):
    child_m = copy.deepcopy(child)
    leyers =  ['W1','W2', 'W3']
    chosen_layer_index = random.randrange(0,2)
    chosen_layer = leyers[chosen_layer_index]

    random_weights = np.random.uniform(-1,1)

    #get x an y from shape
    x_max, y_max = child[chosen_layer].shape

    # index to mutate
    x = random.randrange(0,x_max-1)
    y = random.randrange(0,y_max-1)

    child_m[chosen_layer][x][y] = random_weights

    return child_m

  def tournament(self, fitness_population):
```

```
    f1 = random.randint(0, len(fitness_population) - 1)
    f2 = random.randint(0, len(fitness_population) - 1)
    fit1 =fitness_population[f1]['fitness'] ,
    ch1 = fitness_population[f1]['individual']
    fit2 =fitness_population[f2]['fitness'] ,
    ch2 = fitness_population[f2]['individual']

    return ch1 if fit1 > fit2 else ch2

  def generate_weights_individu(self, sizes):

    # number of nodes in each layer
    input_layer=sizes[0]
    hidden_1=sizes[1]
    hidden_2=sizes[2]
    output_layer=sizes[3]

    params = {
      'W1':np.random.uniform(-10,10, size=(hidden_1,input_layer)),
      'W2':np.random.uniform(-10,10, size=(hidden_2, hidden_1)) ,
      'W3':np.random.uniform(-10,10, size=(output_layer, hidden_2))
    }

    return params

x = df1.drop('Result', inplace=False, axis=1) #remove 'target' column from input
features
y = df1['Result'] #stores target (1 or 0) in a separate array

from sklearn.model_selection import train_test_split

X_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.1,
random_state = 0)

X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

X_train = np.array(X_train, dtype=float)
y_train = np.array(y_train, dtype=float)

y_train = np.array(y_train)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV

# Define the grid of values
penalty = ['l1', 'l2']
C = [0.8, 0.9, 1.0]
tol = [0.01, 0.001 ,0.0001]
max_iter = [100, 150, 200, 250]
```

```
# Create a dictionary where tol and max_iter are keys and the lists of their
values are the corresponding values
param_grid = dict(penalty=penalty, C=C, tol=tol, max_iter=max_iter)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

random_model = RandomizedSearchCV(estimator=logreg,
param_distributions=param_grid, cv=5)

# Fit random_model to the data
random_model_result = random_model.fit(X_train, y_train)

# Summarize results
best_score, best_params = random_model_result.best_score_,
random_model_result.best_params_
print("Best score: %.2f using %s" % (best_score*100., best_params))

# gradeFunction, maxLoopsNum, error, populationSize, crossThreshold,
mutationThreshold
maxLoopsNum = 200
error = 0.95
populationSize = 100
crossThreshold = 0.8
mutationThreshold =  0.01

ag =  AG(rules(maxLoopsNum, error,  populationSize, crossThreshold,
mutationThreshold, X_train, y_train,[30, 50, 2, 1]))
ag.run()

x_test = np.array(x_test, dtype=float)
y_test = np.array(y_test, dtype=float)

y_test = np.array(y_test)

weights =  ag.get_best_individua()
nn = NeuralNetwork(sizes=[30, 10, 5, 1], weights = weights)
print(nn.prediction_score(x_test,y_test))
```

CHAPTER 6

REFERENCES

[1] Yazan Ahmad Alsariera; Victor Elijah Adeyemo; Abdullateef Oluwagbemiga Balogun; Ammar Kareem Alazzawi; “AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites” IEEE Access(Volume: 8) pp. 142532-142542

[2] Rizka Widyarini Purwanto; Arindam Pal; Alan Blair; Sanjay Jha; “PhishSim: Aiding Phishing Website Detection With a Feature-Free Tool” IEEE Transactions on Information Forensics and Security (Volume: 17) pp.1497-1512

[3] Lizhen Tang; Qusay H. Mahmoud; “A Deep Learning-Based Framework for Phishing Website Detection” IEEE Access (Volume: 10) pp.1509-1521

[4] Lakshmana Rao Kalabarige; Routhu Srinivasa Rao; Ajith Abraham; Lubna Abdelkareim Gabralla; “Multilayer Stacked Ensemble Learning Model to Detect Phishing Websites” IEEE Access (Volume: 10) pp. 79543 – 79552

[5] Peng Yang; Guangzhen Zhao; Peng Zeng; “Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning” IEEE Access (Volume: 7) pp. 15196 - 15209

[6] Saad Al-Ahmadi; Afrah Alotaibi; Omar Alsaleh; “PDGAN: Phishing Detection With Generative Adversarial Networks” IEEE Access (Volume: 10) pp. 42459 - 42468

[7] Waleed Ali; Sharaf Malebary; “Particle Swarm Optimization-Based Feature Weighting for Improving Intelligent Phishing Website Detection” IEEE Access (Volume: 8) pp. 116766 - 116780

[8] Yi Wei; Yuji Sekiya; “Sufficiency of Ensemble Machine Learning Methods for Phishing Websites Detection” IEEE Access (Volume: 10) pp. 124103 – 124113

[9] Ilker Kara; Murathan Ok; Ahmet Ozaday; “Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites With Machine Learning Methods” IEEE Access (Volume: 10) pp. 124420 – 124428

[10] Subhash Ariyadasa; Shantha Fernando; Subha Fernando; “Combining Long-Term Recurrent Convolutional and Graph Convolutional Networks to Detect Phishing Sites Using URL and HTML” IEEE Access (Volume: 10) pp. 82355 – 82375

[11] https://www.kdnuggets.com/2022/07/logistic-regression-work.html#:~:text=is%20Logistic%20Regression%3F-,Logistic%20regression%20is%20a%20Machine%20Learning%20classification%20algorithm%20that%20is,the%20logistic%20of%20the%20result.

[12] https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[13] https://www.kdnuggets.com/2020/01/random-forest-powerful-ensemble-learning-algorithm.html

[14] https://www.kdnuggets.com/2020/03/machine-learning-algorithm-svm-explained.html

[15] https://www.kdnuggets.com/2022/04/nearest-neighbors-classification.html

[16] https://www.kdnuggets.com/2019/04/naive-bayes-baseline-model-machine-learning-classification-performance.html

[17] https://www.kdnuggets.com/2016/11/quick-introduction-neural-networks.html

[18] https://www.kdnuggets.com/2018/04/genetic-algorithm-key-terms-explained.html#:~:text=A%20genetic%20algorithm%20(GA)%20characterizes,predefined%20numerical%20measure%2C%20or%20fitness.(Reference)

[19] https://medium.com/@nansha3120/bernoulli-naive-bayes-and-its-implementation-cca33ccb8d2e#:~:text=To%20understand%20it%20in%20layman's,binary%20valued%20variable%20i.e.%20boolean.