VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering (SITE)

B.Tech (Information Technology)

Course Project Report

Phishing Link Detection using Cloud Computing

Submitted for the Course:
ITE 3007 : Cloud Computing and Virtualization

Offered by Dr. R. K. NADESH during SUMMER SEMESTER TERM 5 - 2022

By

Kunj Patel	**19BIT0256**
Tanishq	**19BIT0275**
Yash Sen	**19BIT0266**

AUGUST 2022

School of Information Technology & Engineering
B.Tech (Information Technology)
SUMMER SEMESTER TERM 5 - 2022
ITE 3007 : Cloud Computing and Virtualization

A Report on the Course Project
Phishing Link Detection using Cloud Computing

TEAM Name : Cloud Jumpers
Team Member(s) with Reg # and Name : Kunj Patel; 19BIT0256; 9409254293; kunj.patel2019@vitstudent.ac.in Tanishq; 19BIT0275; 8178529506; tanishq2019@vitstudent.ac.in Yash Sen; 19BIT0266; 9519265953; yash.sen2019@vitstudent.ac.in
Project Title : Phishing Link Detection using Cloud Computing

1. Problem Statement

1.1 Background (System Study Details in brief)

Phishers employ websites that resemble those genuine websites both aesthetically and linguistically. As a result, we create phishing URL detection websites and distribute them in the cloud so that many people may utilize them. This allows users to determine if a URL is phishing or not before clicking on it.

1.2 Problem Statement

The internet has integrated seamlessly into our daily lives, but it has also made it possible to carry out harmful acts like phishing invisibly. Phishers attempt to trick their victims by employing social engineering techniques or building fake websites in order to obtain information such as account IDs, usernames, and passwords from people and businesses. Although several strategies have been put out to identify phishing websites, phishers have developed ways to circumvent these strategies. Machine learning is one of the best techniques for spotting these dangerous behaviors. This is so that machine learning techniques can recognise the common traits shared by the majority of phishing assaults.

1.3 Novelty

The presented system proposes developing a model for detecting phishing links. This is done by using various machine learning and neural network algorithms. For this we have used a dataset of around 40k entries of both types of links, legitimate and the phishing link. Then conducting data preprocessing on the dataset and visualizing it. After splitting the data, we train various models and measure the accuracy of each model. The model which provides the best accuracy is selected for final deployment. This model is then used to evaluate the input link given by the user. Percentage is displayed to display whether the link is phishing or not, and if so by what percentage. To do a website template has been used to get the input from the user.

1.4 Dataset

The dataset is imported from the kaggle website named "Phishing Link detector". The overview of this dataset is that it has 11054 samples (rows) and 32 features (columns).
Link For Dataset:
https://drive.google.com/drive/folders/1DYMDzPid02ZnEzKe6EQhzsJsj5ffmwxe?usp=sharing

2. Related Works

2.1 Literature Survey (Should be elaborately discussed with its citation)

1. Phishing Website Detection Based on Machine Learning: A Survey

In the modern world, phishing attacks are cybercrimes carried out via software and social engineering. One of most serious risks that every person and organization faced was this one. Users find information on the internet via URLs, often known as web links. The review advises readers to exercise phishing protection by raising their knowledge of phishing assaults, helping them identify phishing attempts, and more. In phishing, phishers utilize email or messages as a tool to target a particular person or group of people by sending them a URL link that will trick them. Companies or individuals are unable to identify all the phishing emails or messages that are sent out every day due to the sheer volume of them that are received. According to APWG, 1,80,768 phishing sites were found in the first quarter, and there were 112,393 unique phishing reports, SaaS & Webmail providers were the industry most frequently attacked, with 36% of all phishing attempts documented targeting their brands. The two main categories of phishing attempts are those based on social engineering and malware. Social engineering assaults typically prey on users' minds and susceptibilities to persuade them into disclosing sensitive information. Malware-based phishing is a type of scam in which unauthorized software or programmers are installed on the user's computer. It is a survey, and the machine learning algorithm used a mixture of 30 features to detect with about 99% accuracy. Phishing detection tools let visitors know whether a website is real or suspicious.

2. PhishCatch – A Phishing Detection Tool

The PhishCatch algorithm, which was created to identify phishing, is explained in this document along with the most widely used phishing techniques. The PhishCatch algorithm, which is based on heuristics, will identify phishing emails, and notify users of them. The algorithm's phishing filters and rules were developed after a thorough analysis of phishing approaches and techniques. The PhishCatch algorithm was shown to have a catch rate of 80% and an accuracy of 99% after testing. This paper discusses the methodology used to build the algorithm, the implementation details, and the testing outcomes. The core components of PhishCatch's design include a module for retrieving emails, a module for filtering emails and classifying them as phishing, an alerter for users, and a data warehouse for storing all the information about phishing emails. The new emails are initially fetched by the PhishCatch algorithm from the SMTP server. POP and IMAP mail servers are compatible with the algorithm's architecture. Any new email that arrives is retrieved and divided into the headers and body. In some cases, email bodies are HTML-encoded, and the "Content-type" field in the email header makes this information clear. If the email is encrypted, we decode it to make sure the phishing filters can properly process it. The next stage in the algorithm is to add the phishing filters to the email to identify a phishing email after it has been recovered and broken down into its component pieces. PhishCatch is an algorithm that focuses on identifying phishing connections, warning users about potential phishing links, and constructing a sizable data warehouse with a wealth of phishing-related knowledge. This dataset can also be used to generate statistics regarding phishing and examine patterns in phishing. PhishCatch is an extremely efficient method with a capture rate of 80% and accuracy of 99%. It uses very little memory and CPU time.

3. ANTI-PHISHING DETECTION OF PHISHING ATTACKS USING GENETIC ALGORITHM

A method for phishing hyperlink identification utilizing a rule-based system created by a genetic algorithm is suggested. This method can be used as a component of an enterprise anti-phishing solution. This method of online search can be used by a lawful website owner to look for questionable links. In this method, rules are developed using a genetic algorithm to distinguish between authentic and phishing links. GA creates a ruleset that only matches phishing URLs by evaluating parameters such as evaluation function, crossover, and mutation. This ruleset is kept in a database, and if a link meets any of the criteria in the rule-based system, it is flagged as a phishing link, protecting users from pretend hackers. Initial tests demonstrate that this method effectively detects phishing hyperlinks with a low false negative rate at a fast enough rate for online application. A set of computational models known as genetic algorithms is based on the concepts of natural selection and evolution. These algorithms use a chromosome-like data structure to model the problem in a particular domain and update the chromosomes using mutation, recombination, and selection operators. Simple guidelines for thwarting phishing attacks can be evolved using genetic algorithms. These guidelines are used to distinguish between typical websites and aberrant websites. These strange websites refer to things that could be phishing scams. This rule can be described as follows: phishing emails are classified as such if the Internet protocol of the URL in the email does not match the specified Rule Set for Whitelist. Applying GA ultimately aims to produce rules that only match websites' aberrant URLs. To discover suspicious phishing assaults, these rules filter new URLs after being checked on older URLs. The hyperlinks that have been contained in phishing emails are highlighted in this document. The Genetic Algorithm then creates a ruleset that is used to identify the hyperlink. Before reading the email and providing their information, this added function alerts the user to the mail's state. The user is urged not to read the message once it has been identified as a phishing email and, even if they do, not to click the offered links. The user can access the message that was sent from the originating domain and has the status "Not-Phishing."

4. <u>**Online Detection and Prevention of Phishing Attacks**</u>

This study uses the common traits of hyperlinks used in phishing attempts to propose a novel end-host based anti-phishing solution that we call LinkGuard. These features were discovered through analysis of the Anti-Phishing Working Group's archive of phishing data (APWG). LinkGuard can identify both known and undiscovered phishing attacks because it is based on the general features of these attacks. LinkGuard has been added to Windows XP. Our tests showed that LinkGuard effectively detects and blocks both known and unidentified phishing attacks with a low rate of false negatives. Out of the 203 phishing assaults, LinkGuard correctly identifies 195. The tests also showed that LinkGuard is lightweight and capable of real-time phishing attack detection and prevention. Analysis of the discrepancies between the apparent link and the genuine link is how LinkGuard operates. Additionally, it determines how closely a URI resembles a well-known, reputable website. First, it pulls the DNS names from the real and visual connections in its core process, LinkGuard. After that, it analyses the visual and actual DNS names; if they differ, it is category 1 phishing. Direct use of dotted decimal IP addresses in actualhdns indicates a potential category 2 phishing attempt. Analysis of the discrepancies between the apparent link and the genuine link is how LinkGuard operates. Additionally, it determines how closely a URI resembles a well-known, reputable website. First, it pulls the DNS names from the real and visual connections in its core process, LinkGuard. After that, it analyses the visual and actual DNS names; if they differ, it is category 1 phishing. Direct use of dotted decimal IP addresses in actualhdns indicates a potential category 2 phishing attempt. Since Phishing- Guard is based on characteristics, it is effective against both known and undiscovered attacks. For Windows XP, we put LinkGuard into practice. Our test demonstrated that LinkGuard is lightweight and capable of quickly identifying up to 96% of unidentified phishing attacks. It is thought that LinkGuard can protect users from harmful or uninvited links in Web sites and Instant messaging in addition to being helpful for identifying phishing assaults.

5. <u>**Phishing Email Detection Technique by using Hybrid Features**</u>

This study suggests using hybrid traits to recognize phishing emails. Content-based, URL-based, and behavior-based features make up the hybrid features. The proposed method produced an overall accuracy of 97.25% and an error rate of 2.75% based on a batch of 500 phishing emails and 500 genuine emails. This encouraging outcome supports the efficiency of the suggested hybrid features in phishing email detection. Features and four features based on sender behavior. Blacklist terms in subject and content, domain sender, URL IP address, URL dots, and URL symbol are all email structure-based features. Unique recipient, unique domain, return path, and hyperlink are included in sender behavior-based features. The numerical features of a URL include its IP address, dot, and symbol. The vast range of potential values for numerical attributes is a concern. Therefore, before the classification procedure, the numerical characteristic needs to be normalized. So that all proposed traits can be seen as equally relevant, the numerical values are scaled to the range of [0,1]. The product desirable over the feature's maximum value will be used to normalize each feature with quantitative data. In this study, SVM was selected as the classifier. The suggested method outperforms the HA method and yields encouraging results. It is important to highlight that the quality and amount of the corpus depend on the efficiency of the unique responder and unique domain properties. As a result, it can take longer. Due to the lack of an updated banned terms list, there are some restrictions on the blacklist keywords feature. The email's graphical element, which is in HTML format, is not considered by the current implementation. This component is also quite significant.

6. <u>Phishing Emails Detection Using CS-SVM</u>

High accuracy phishing email detection has drawn a lot of attention. Support Vector Machine (SVM) is one of the machine learning-based detection techniques that has been shown to be efficient. However, the kernel method's parameters, whose default is to classify integers as reciprocals generally, have an impact on the SVM's classification accuracy. In this research, a model called Cuckoo Search SVM is proposed to increase classification accuracy (CS-SVM). 23 features are extracted by the CS-SVM and used to build the hybrid classifier. Cuckoo Search (CS) is linked with SVM within hybrid classifiers to maximize Radial Basis Function parameter selection (RBF). The based scheme classifier focuses on feature extraction and classifier development for the detection of phishing emails. Fig. 1 depicts the proposed method's architecture. In the pre-processing stage, emails are converted to XML format and divided into three parts: the header, the body, and the URL. The second stage oversees feature standardization and feature extraction depending on the email's header, body, and URL. Then, using CS-SVM as our classifier to detect phishing emails, we optimize the kernel function parameter in SVM using CS. The best features of the Cuckoo search algorithm and the SVM classifier are combined in this study to choose the best parameter value for the kernel function. In addition, we choose 23 features, such as content, URL, and header. We run tests on a dataset that consists of three archives. According to experimental findings, CS-SVM has a greater phishing email detection result at various training sets. This shows that the suggested approach outperforms the SVM classifier's default parameter value.

7. <u>Phishing Websites Detection Using Machine Learning</u>

Phishing website URLs are designed to steal personal data, including usernames, passwords, and online financial activities. Phishers employ websites which resemble those genuine websites both aesthetically and linguistically. Utilizing anti-phishing methods to identify phishing is necessary to stop the rapid advancement of phishing techniques because of advancing technology. A strong tool for thwarting phishing assaults is machine learning. The characteristics and machine learning-based detection methods are surveyed in this work. This study suggests utilizing a c4.5 decision tree technique to identify phishing URL domains. This method determines heuristic values by extracting features from the sites. The c4.5 decision tree algorithm was given these values to calculate if the site is phishing or not. Dataset is gathered from Google and Phish Tank. The pre-processing phase and the detecting phase are both parts of this procedure. Features are retrieved using pre-processing criteria, and the elements and their corresponding values are then fed into the c4.5 algorithm, which yields an accuracy of 89.40%. This study focuses on identifying domain name elements in phishing website URLs. The three types of web spoofing attacks—content-based, heuristic-based, and blacklist-based—are described, and the suggested model, PhishChecker, is created using Microsoft Visual Studio Express 2013 and the C# programming language. The Phishtank and Yahoo directory data sets were used, and the accuracy was 96%. This study only examines the reliability of URLs. This study provided multiple machine learning researchers' algorithms and methods for identifying phishing websites. After reading the articles, we deduced that most of the work was carried out using well-known machine learning techniques such as Naive Bayesian, SVM, Decision Tree, and Random Forest. Some authors suggested a brand-new approach for detection, like PhishScore and PhishChecker. In terms of accuracy, precision, recall, etc., feature combinations were used. Phishing websites are becoming more prevalent every day, thus elements that used to identify them may be added or substituted with new ones.

8. <u>A machine learning based approach for phishing detection using hyperlinks information</u>

This study proposes a novel method for identifying phishing attacks by examining the website's HTML source code for hyperlinks. The suggested method includes several fresh, noteworthy hyperlink-specific properties to identify phishing attacks. To train the machine-learning algorithms, the proposed approach separated the hyperlink-specific attributes into 12 different groups. With the help of the dataset of phishing and non-phishing websites, we assessed how well our suggested phishing detection strategy performed against different categorization algorithms. The suggested method is totally client-side and does not call for any third-party services. The suggested method also detects websites published in any text language and is language independent. The proposed methodology has a reasonably high accuracy in detecting phishing websites when compared to previous approaches because it obtained more than 98.4% accuracy on the logistic regression classifier. The main contribution of this study is the identification of an excellent feature set. Six new features have been suggested by Paper to increase the effectiveness of phishing webpage detection. The relationship between the site material and the URL is identified by our suggested features. The website's hyperlinks form the foundation of our offerings. To extract the linking features from a website, a Document Object Model (DOM) tree can be created. The hierarchically known absolute connections are used in place of the relative links. Our suggested method bases its decision on 12 criteria, including the total number of hyperlinks, the absence of hyperlinks, both internal and external hyperlinks, null hyperlinks, internal and external errors, internal and external redirects, login form links, external/internal CSS, and external/internal favicons. This study identified a few novel characteristics that can be used to spot phishing websites. These functions are based on hyperlink information provided in the website's source code. These characteristics were used by the model to develop a logistic regression classifier that was highly accurate at distinguishing between real and phishing websites. The choice of hyperlink-specific properties, which are derived from client side and do not rely on any third-party services, is one of the paper's significant contributions. Additionally, these qualities are adequate to identify a website that is written in any language. The testing findings demonstrated that the suggested method is quite effective at classifying phishing websites, with a true positive rate of 98.39% and an overall accuracy of 98.42%.

9. <u>Phishing Website Detection based on Multidimensional Features driven by Deep Learning</u>

In this study, we offer a rapid deep learning-based multiple feature phishing detection solution (MFPD). The given URL's character sequence features are extracted in the first phase and used by deep learning for rapid classification; this step doesn't need outside help or any prior phishing knowledge. In the second stage, we mix deep learning's rapid classification output with URL statistical data, webpage code characteristics, webpage text features, and multidimensional features. The method can decrease the amount of time needed to detect something before setting a threshold. Test results on a dataset with millions of legitimate and phishing URLs show that the accuracy is 98.99% and the rate of false positives is only 0.59%. By detecting or zero-filling, the URL characters sequence is normalized to a fixed-size sequence before being transformed into a one-hot code sequence in accordance with Table 1. The embedding layer then transforms the sparse one-hot matrix into a dense character embedding matrix. The local deep correlation feature is produced from the embedding matrix during the feature extraction stage using the convolutional layer and maximal CNN pooling. The LSTM neural network is then fed the pooling result to determine the context of the URL sequence. The output of the LSTM neural network's last instant is sent into the SoftMax unit during the classification step. A dropout approach is employed to avoid overfitting, and SoftMax outputs the likelihood that the URL leads to a phishing website. This viewpoint is supported by the MFPD technique that has been suggested. The URL character sequence without prior knowledge of phishing provides the speed of detection, and the multidimensional feature detection ensures the accuracy of detection, both under the direction of a dynamic category selection algorithm. On a dataset including millions of authentic and phishing URLs, several tests were run. We conclude from the results that the MFPD strategy is successful due to its high accuracy, low false positive rate, and rapid detection.

10. <u>HTMLPhish: Enabling Phishing Web Page Detection by Applying Deep Learning Techniques on HTML Analysis</u>

This study suggests HTMLPhish, a data-driven end-to-end autonomous phishing web page categorization solution based on deep learning. HTMLPhish uses convolutional neural networks (CNNs) to discover the semantic dependencies in the textual contents of the HTML document after receiving the content of the HTML document from a web page. Without considerable human feature engineering, the CNNs learn appropriate feature interpretations from the HTML page embeddings. Additionally, the suggested method of joining the word and character embeddings enables our model to handle novel features and ensures straightforward projection to test data. An extensive study that used a dataset of much more than 50,000 HTML documents produced results with a true positive rate and accuracy of over 93%. It provides a dispersion of phishing to safe web pages that are accessible in the actual world. Additionally, HTMLPhish is a client-side method that is totally language-independent, enabling it to undertake web page phishing detection irrespective of the textual language. There are two input layers in the HTMLPhish- Full network architecture. While the second input layer processes words, the first input layer turns the raw HTML content into an embedding matrix made up of character-level feature representations. In a thick layer known as the Concatenation layer, these two branches are joined together. The HTML content of a web page is inputted into HTMLPhish, which then uses CNNs to create a jointly optimal network to learn the semantic connections between the characters and words in the HTML document. In addition, we used convolutions on a combination of the character and word embedding matrices to make sure that new words were effectively inserted into the test HTML texts. With the suggested method, context features can be learned from HTML pages without the need for a lot of human feature engineering

11. <u>Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions</u>

Phishing has drawn the attention of security professionals as well as end users due to its growing concern. Despite years of development and refinement, current phishing detection algorithms still have poor performance accuracy and cannot detect unidentified attacks. This study uses a systematic literature review approach to analyse 81 chosen papers in order to offer a taxonomy of deep learning algorithms for phishing detection. The paper begins by discussing deep learning and phishing in the context of cybersecurity. In order to address these hurdles, the paper highlights several difficulties deep learning encounters while phishing detection and suggests future research possibilities. Finally, an empirical analysis is carried out to assess how well various deep learning algorithms function in a practical setting and to highlight the essential concerns that will inspire academics to continue their work in the future. According to the empirical experiment's findings, manual parameter adjustment, protracted training periods, and poor detection accuracy are prevalent problems with modern deep learning systems.

12. <u>Detecting Cloud-Based Phishing Attacks by Combining Deep Learning Models</u>

Nowadays, major cloud site hosting services and applications like Google Sites and Typeform are used by web-based phishing attempts to host their attacks. Traditional phishing detection techniques like IP reputation monitoring and blacklisting are not particularly successful because these assaults come from trustworthy sites and IP addresses of the cloud services. Here, we look into how well deep learning models can identify this particular subset of cloud-based phishing attacks. We specifically assess deep learning models for three phishing detection techniques: triplet network model for visual similarity analysis, YOLOv2 model for logo analysis, and LSTM model for URL analysis. We use well-known datasets to train the models, then we assess their effectiveness against actual phishing attacks. Our findings provide a qualitative analysis of the models' success or failure.

13. <u>Evolutionary Algorithm with Deep Auto Encoder Network Based Website Phishing Detection and Classification</u>

For the purpose of drawing a sizable number of Internet visitors, the phishing websites closely resemble their comparable authentic websites. By presenting the masked webpage as real or trustworthy for retrieving its crucial information, the attacker dupes the user. Currently, anti-phishing strategies require professionals to identify the characteristics of phishing websites and use third-party services to identify phishing websites. These methods have considerable limitations because it takes time to extract phishing elements without the help of professionals. Blacklist or whitelist, heuristics, and machine learning (ML) based approaches are just a few of the solutions for phishing website attacks that have been proposed. However, these approaches struggle to achieve effective recognition performance due to the ongoing development of phishing technologies. This study proposes the best website phishing detection and classification (ODAE-WPDC) model based on deep autoencoder networks. The initial stage of the proposed ODAE-WPDC model employs input data pre-processing to remove any missing values from the dataset. Then, feature extraction and feature selection based on the artificial algae algorithm (AAA) are used. The experimental results support the ODAE-WPDC model's superior performance, which has a maximum accuracy of 99.28%.

14. An Effective and Secure Mechanism for Phishing Attacks Using a Machine Learning Approach

One of the biggest crimes in the world, phishing entails the stealing of the user's private information. Phishing websites frequently target the websites of businesses, organizations, governments, and cloud storage providers. When using the internet, the majority of individuals are not aware of phishing assaults. Numerous phishing techniques now in use don't effectively address the problems caused by email attacks. To combat software attacks, hardware-based phishing techniques are now deployed. The proposed effort concentrated on a three-stage phishing series attack for precisely detecting the difficulties in a content-based manner due to the rise in these types of problems. To implement the proposed phishing attack mechanism, a dataset is collected from recent phishing cases. It was found that real phishing cases give a higher accuracy on both zero-day phishing attacks and in phishing attack detection. Three different classifiers were used to determine classification accuracy in detecting phishing, resulting in a classification accuracy of 95.18%, 85.45%, and 78.89%, for NN, SVM, and RF, respectively. The results suggest that a machine learning approach is best for detecting phishing.

15. A novel Technique: Data Leakage Hindering in Cloud computing using Swarm Intelligence

The primary goal of this context is to draw attention to the problem of data security and privacy, which may result in data leakages in cloud services. This research introduces a revolutionary cloud computing strategy for impeding data leakage. This cutting-edge method involves preventing data leakage caused by interruptions in the transmission line. Rather than detecting or stopping the interruption, it would lead to a careful path that would stop all data leakage. In order to block the communication line that is broken or where data is stolen, this work aims to integrate the ant colony optimization technique (ACO) and the artificial bee colony technique (ABC). The ABC creates a likely and careful way to the server, and the ACO responds for the required path. If a data leakage occurs, the ABC would also back forward the information. The ACO's security controls would be where the data leakage would start. In order to stop the data leakage in the cloud service, the ABC and ACO are combined.

16. Dynamic data leakage detection model based approach for MapReduce computational security in cloud

The suggested methodology divides the whole amount of data into manageable portions, increasing parallel processing and reducing execution time. Then, using a well-known Hadoop framework to analyse, filter, and reduce the data, the entire set of data is made to go through the map-reduce process. Map Reduce is mostly composed of Job Follower and Task Follower. These components will further distribute the jobs to slave nodes. Up to 70% less data must be stored as a result.

17. Towards Cooperative Perception Services for ITS: Digital Twin in the Automotive Edge Cloud

a practical working prototype of a cooperative perception system that keeps a real-time digital twin of the traffic environment and offers a more precise and dependable model than any of the participant subsystems—in this case, smart vehicles and infrastructure stations—would manage separately. Such technology is significant because it can enable a variety of new auxiliary services, such as cloud-assisted and cloud-controlled ADAS functions, dynamic map generation with analytics for traffic control and road infrastructure monitoring, a digital framework for operating vehicle testing grounds, logistics facilities, etc. Implement a system that only offers one service: the live depiction of our digital twin in a 3D simulation, which accurately and quickly reflects the status of the real-world setting and demonstrates the benefits of real-time fusion of sensory data from various traffic participants. We see this prototype system as a component of a wider network of regional information processing and integration nodes, where the physically scattered edge cloud serves as the home for the conceptually centralised digital twin.

18. Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster

More businesses are now moving toward embracing cloud environments to offer their clients services. Although setting up a cloud environment is simple, it's equally crucial to monitor the system's behaviour during runtime and spot any unusual behaviours that emerge. The use of "acrnn" and "acdnn" to find abnormalities that could happen while running has become popular. An easier way to grasp and comprehend the behaviour that state machine models model is provided. In this work, we offer a method for simulating the behaviour of a cloud environment running several microservice applications by learning state machine models. The attempt to use state machine models with microservice architectures is made for the first time in this work. We launch numerous forms of attacks on the cloud environment, and the state machine model is utilised to identify them. According to the findings of our trial, our method is quite effective in identifying attacks, with a balanced accuracy of 99.2% and an F1 score of 0.982

19. Outsourcing business to cloud computing services: Opportunities and challenges

The hitherto fantastical idea of starting and operating a virtual business—a company where the majority or all of its business tasks are outsourced to online services—has become more conceivable thanks to developments in service oriented architecture (SOA). The SOA concept is realised through cloud computing, where IT resources are made available as services that are more cost-effective, adaptable, and appealing to enterprises.Improvements in cloud computing, as well as the advantages and trade-offs that enterprises must weigh when employing cloud services. A conceptual architecture for a virtual business operating environment, and a layered architecture for the virtual business. The potential and effects of cloud services for both big enterprises and small ones.

20. A novel Technique: Data Leakage Hindering in Cloud computing using Swarm Intelligence

The primary goal of this context is to draw attention to the problem of data security and privacy, which may result in data leakages in cloud services. This research introduces a revolutionary cloud computing strategy for impeding data leakage. This cutting-edge method involves preventing data leakage caused by interruptions in the transmission line. Rather than detecting or stopping the interruption, it would lead to a careful path that would stop all data leakage. In order to block the communication line that is broken or where data is stolen, this work aims to integrate the ant colony optimization technique (ACO) and the artificial bee colony technique (ABC). The ABC creates a likely and careful way to the server, and the ACO responds for the required path. If a data leakage occurs, the ABC would also back forward the information. The ACO's security controls would be where the data leakage would start. In order to stop the data leakage in the cloud service, the ABC and ACO are combined.

21. LARX: Large-scale Anti-phishing by Retrospective Data-Exploring Based on a Cloud Computing Platform

In this research, we present LARX, an offline phishing assault forensics collecting and analysis system, which stands for Large-scale Anti-phishing by Retrospective data-eXploration. First, we gather network trace data via traffic archiving from a vantage point. Second, using a "split and conquer" strategy-like approach to evaluate the experimental data, we make use of cloud computing technologies. Amazon Web Services and Eucalyptus are two currently utilised cloud computing systems. Additionally, a real server is utilised for comparison. The cloud computing platform serves as the foundation for all of LARX's phishing screening activities, which are completely concurrent. Finally, we draw the conclusion that LARX may be successfully scaled up to analyse a sizable number of network trace data for phishing attack detection as an offline solution.

22. Cloud Computing-Based Forensic Analysis for Collaborative Network Security Management System

The design and implementation of a cloud-based security centre for network security forensic investigation are suggested in this research. We suggest storing traffic data obtained using cloud storage and analysing it using cloud computing platforms to identify harmful assaults. As a useful illustration, forensic analysis of a phishing assault is provided, and the necessary computer and storage resources are assessed based on actual trace data. Each cooperative UTM and prober may be given instructions by the cloud-based security centre to gather events and unprocessed traffic, transmit it back for in-depth analysis, and produce new security rules.

23. Hybrid Rule-Based Solution for Phishing URL Detection Using Convolutional Neural Network

37 characteristics from six distinct methods—the black listed technique, the lexical and host method, the content method, the identity method, the identity similarity method, the visual similarity method, and the behavioural approach—are included in the study. Additionally, a comparison of several machine learning and deep learning models, such as MLP (multilayer perceptron) and CNN, as well as CART (decision trees), SVM (support vector machines), and KNN (-nearest neighbours), was conducted (CNN). The study's conclusions showed that the approach was successful in analysing URL stress from various angles, supporting the model's validity. However, deep learning was able to achieve the maximum accuracy level, with reported values for the CNN and MLP models of 97.945 and 93.216, respectively.

24. Cloud based content fetching: Using cloud infrastructure to obfuscate phishing scam analysis

In the current work, we suggest employing a cloud architecture to conceal our identity from phishing sites by deploying virtual computers dispersed over several geographical areas. By employing several IP addresses and surfing setups, our system displays various identities and user behaviour to the phishing sites. We demonstrate the success of our strategy in preventing detection and blockage of anti-phishing probes by the operators of the phishing site by conducting a 10-day probe experiment against a real phishing site

25. NOPHISH: A PHISH DETECTOR IN CLOUD SERVICES

Strong browsing tools should be made available to cloud users by cloud service providers, and these tools must be able to identify phishing websites before they defraud cloud users or service providers. The cloud service provider must offer browser plugins that can identify phishing attempts and block them. Using modified Grey Wolf Optimization (MGWO) for feature selection and Support Vector Machine (SVM) for classification, we have suggested the NoPhish phish detection model in this study. By utilising MGWO, we are able to obtain an optimal feature subset, which significantly lowers the CPU and memory demands during the SVM's training and classification processes. According to the findings of the experiment, the suggested phish detection model has a higher detection accuracy and true positive rate.

26. Advanced Persistent Threat attack detection method in cloud computing based on auto encoder and softmax regression algorithm

Advanced Persistent Threat (APT) is a sophisticated kind of assault that obtains personal data by remaining infected systems for a protracted period of time. It is particularly challenging to identify APT attacks using conventional techniques when they occur in a dynamic and sophisticated infrastructure like the cloud. The study suggests using a deep learning strategy based on autoencoders to detect APT attacks in order to get around the shortcomings of current techniques. The benefit of this model is that it finds intricate connections between features in a database to get a high classification result. Additionally, the model reduces the quantity of data in the encoder, simplifying the classification of enormous quantities of data.

27. Efficient spear-phishing threat detection using hypervisor monitor

In order to combat spear-phishing threats, we design a Cloud-threat Inspection Appliance (CIA) system in this study that focuses on the obstacles encountered. We leverage the CIA to create a transparent hypervisor monitor that hides the existence of the detection engine in the hypervisor kernel, taking use of the benefits of hardware-assisted virtualization technology. To improve system speed, the CIA also develops a document pre-filtering algorithm. The suggested CIA was able to filter 77% of PDF attachments by looking at the structure of the PDF file and preventing them from all being routed to the hypervisor monitor for more in-depth study. Finally, we put CIA to the test in real-world situations. A superior anti-evasion sandbox than commercial ones was demonstrated for the hypervisor monitor.

28. Evolving cloud security technologies for social networks

The development of cloud computing (CC) models has made it possible for different kinds of users to store a lot of data in a flexible and scalable way. Although using the cloud as a service (CaaS) to store the data has several benefits, it has been shown that CaaS suffers from a number of security-related problems. These could encounter difficulties with their technology and software, as well as issues with data security, integrity, and availability in the cloud. Therefore, insecure network traffic encryption across the network may potentially allow private information to impact. In CC, data access can be a significant security-related problem.

29. Cyber-Attacks in Cloud Computing: A Case Study

Given the importance of cloud security, this paper makes an effort to identify the main risk variables that might be crucial in a cloud context. It also draws attention to the different strategies used by the assailants to inflict harm. We have examined the key cyber security publications in order to achieve our goal. Cyberattacks are shown to be sector-specific and to differ greatly depending on the type of industry. Last but not least, we investigated a case study of cyberattacks that have previously happened in the cloud paradigm. By classifying cyberattacks into distributed denial of service and phishing assaults, they were brought to light. The industry and researchers would greatly benefit from this study in their efforts to comprehend the many cloud-specific cyber-attacks.

30. Cloud-based email phishing attack using machine and deep learning algorithm

In a phishing assault, the attacker uses email to deliver spam data, and when you open and read the email, they obtain your data. This study employs various valid and phishing data sizes, looks for fresh emails, and uses various attributes and classification algorithms. The present methods are measured before creating a redesigned dataset. Long short-term memory (LSTM), Naive Bayes, and support vector machine (SVM) algorithms were used to build a feature extracted comma-separated values (CSV) file and label file. The comparison and implementation show that SVM, NB, and LSTM perform better and more accurately in identifying email phishing assaults. With accuracy rates of 99.62%, 97%, and 98%, respectively, SVM, NB, and LSTM classifiers are used to categorize email threats.

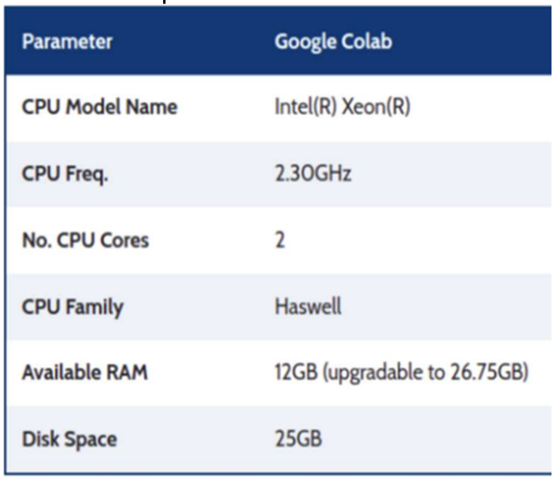
2.2 Comparative statement (10 latest Journal papers in the current domain, Tabulation only)

Research paper	Models used	Results
Cloud-based email phishing attack using machine and deep learning algorithm	Support Vector Machine (SVM), Naive Bayes (NB), LSTM	Accuracy : SVM 99.62%, NB 97%, LSTM 98%
Hybrid Rule-Based Solution for Phishing URL Detection Using Convolutional Neural Network	CART, SVM or KNN, Multi Layer perceptron, CNN	Accuracy : CNN 97.945%, MLP 93.216%

Cloud based content fetching: Using cloud infrastructure to obfuscate phishing scam analysis	Multiple cloud providers such as Amazon EC2, Rackspace and GoGrid	Based on instance type, Amazon EU 15%, Rackspace 20%, GoGrid East 19%, AT&T Client 19%, Amazon East 17%, Amazon Asia 15%
Dynamic data leakage detection model based approach for MapReduce computational security in cloud	Load Balancing, Map Reduce Model, Securing map computations	Load Balancing shows assigning equal data to each VMs, Response Time per machine, evaluation of extent to which data is reduced
NOPHISH: A PHISH DETECTOR IN CLOUD SERVICES	NoPhish model using modified Grey Wolf Optimization (MGWO) for feature selection, SVM for classification	Better detection accuracy and true positive rate
Phishing Email Detection Technique by using Hybrid Features	SVM was selected as the classifier. The suggested method outperforms the HA method and yields encouraging results.	Accuracy: 97.25% Error: 2.75%
Phishing Websites Detection Using Machine Learning	c4.5 decision tree technique to identify phishing URL domains Dataset is gathered from Google and Phish Tank	89.40%
Evolutionary Algorithm with Deep Auto Encoder Network Based Website Phishing Detection and Classification	This study proposes the best website phishing detection and classification (ODAE-WPDC) model based on deep autoencoder networks. Feature extraction and feature selection based on the artificial algae algorithm (AAA) are used	accuracy: 99.28%
An Effective and Secure Mechanism for Phishing Attacks Using a Machine Learning Approach	Three different classifiers were used to determine classification accuracy in detecting phishing, resulting in a classification. NN, SVM, and RF. The results suggest that a machine learning approach is best for detecting phishing.	accuracy of 95.18% NN, 85.45% SVM, and 78.89% RF,
HTMLPhish: Enabling Phishing Web Page Detection by Applying	HTMLPhish uses convolutional neural networks (CNNs) to	An extensive study that used a dataset of much more than

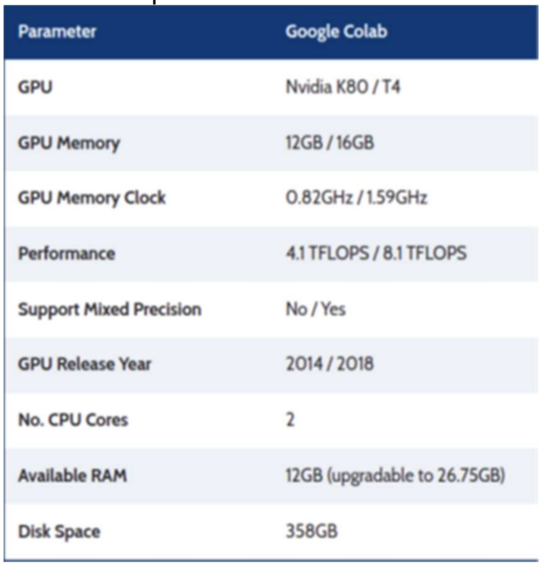
Deep Learning Techniques on HTML Analysis	discover the semantic dependencies in the textual contents of the HTML document after receiving the content of the HTML document from a web page	50,000 HTML documents produced results with a true positive rate and accuracy of over 93%.

2.3 Hardware Requirements



Parameter	Google Colab
CPU Model Name	Intel(R) Xeon(R)
CPU Freq.	2.30GHz
No. CPU Cores	2
CPU Family	Haswell
Available RAM	12GB (upgradable to 26.75GB)
Disk Space	25GB

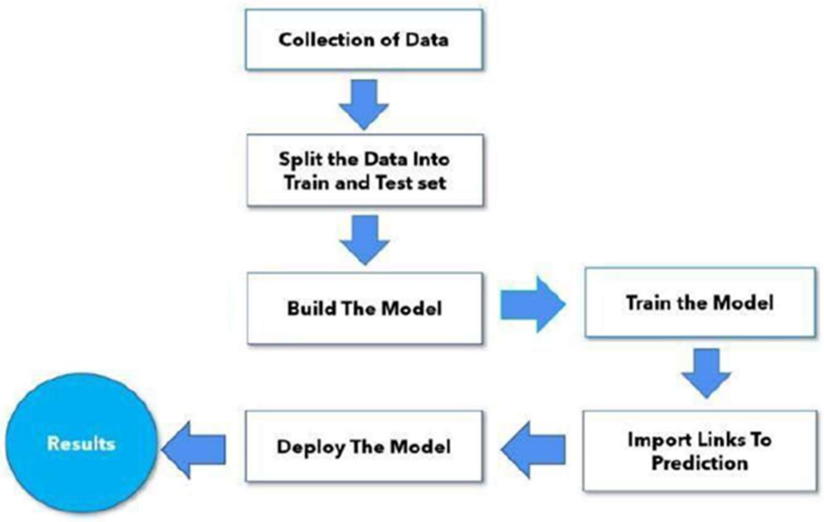
2.4 Software Requirements



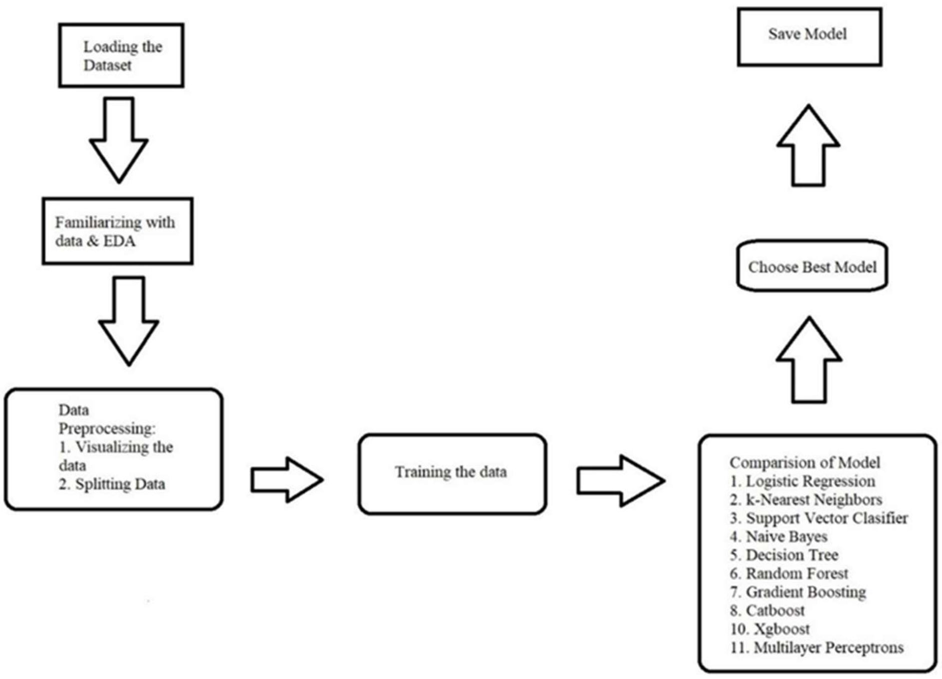
Parameter	Google Colab
GPU	Nvidia K80 / T4
GPU Memory	12GB / 16GB
GPU Memory Clock	0.82GHz / 1.59GHz
Performance	4.1 TFLOPS / 8.1 TFLOPS
Support Mixed Precision	No / Yes
GPU Release Year	2014 / 2018
No. CPU Cores	2
Available RAM	12GB (upgradable to 26.75GB)
Disk Space	358GB

3. System Design

3.1 High-Level Design (Black Box design)



3.2 Low-Level Design (Detailed design)



Both should be drawn and each component should be explained with respect to the proposed problem statement

4. System Implementation

4.1 Algorithms (followed, proposed or altered)

1. **Logistic Regression**
   In a categorical dependent variable, the output is predicted via logistic regression. As a result, the result must be a discrete or categorical value. With the exception of how they are applied, logistic regression and linear regression are very similar. While logistic regression is used to solve classification difficulties, linear regression is used to solve regression problems.
2. **K-Nearest Neighbors**
   One of the simplest machine learning algorithms, based on the supervised learning method, is K-Nearest Neighbour. The K-NN method makes the assumption that the new case and the existing cases are comparable, and it places the new instance in the category that is most like the existing categories.
3. **Support Vector Machine**
   One of the most well-liked supervised learning algorithms, Support Vector Machine, or SVM, is used to solve Classification and Regression issues. The SVM algorithm's objective is to establish the optimal line or decision boundary that can divide n-dimensional space into classes, allowing us to quickly classify fresh data points in the future.
4. **Naive Bayes**
   The Naive Bayes algorithm is a supervised learning method for classification issues that is based on the Bayes theorem. It is mostly used for text and picture classification with a large training set. The Naive Bayes Classifier is one of the most straightforward and efficient classification algorithms available today. It aids in the development of rapid machine learning models capable of making accurate predictions.
5. **Decision Trees**
   A supervised learning method called a decision tree may be used to solve classification and regression issues, but it is often favoured for doing so. It is a tree-structured classifier, where internal nodes stand in for a dataset's characteristics, branches for the decision-making process, and each leaf node for the classification result.
6. **Random Forest**
   Popular machine learning algorithm Random Forest is a part of the supervised learning methodology. It may be applied to ML issues involving both classification and regression. It is predicated on the idea of ensemble learning, which is the act of integrating many classifiers to address a complicated issue and enhance the model's performance.
7. **Gradient Boosting Classifier**
   A family of machine learning techniques known as gradient boosting classifiers combines a number of weak learning models to produce a powerful predicting model. Gradient boosting frequently makes use of decision trees. Regarding the bias variance trade-off, boosting methods are essential. Boosting algorithms are thought to be more successful than bagging algorithms since they regulate both bias and variance in a model, as opposed to bagging algorithms, which solely correct for excessive variance.
8. **CatBoost Classifier**
   Yandex's CatBoost is a recently released machine learning algorithm. It is simple to interface with deep learning frameworks such as Apple's Core ML and Google's TensorFlow. It can operate with various data formats to assist in resolving a variety of issues that organizations are now facing.
9. **XGBoost Classifier**
   Gradient-boosted decision trees are implemented using XGBoost, which is a dominant competitive machine learning method built for speed and performance. This article will show you how to set up and create your first Python XGBoost model.
10. **Multi-layer Perceptron Classifier**
    Multi-layer Perceptron classifier, or MLPClassifier, is connected to a neural network by the term itself. MLPClassifier, in contrast to other classification algorithms like Support Vectors or Naive Bayes Classifier, uses an underlying Neural Network to carry out the classification process.

4.2 Mathematical Model (followed, proposed or altered)

Gradient Boost

Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. for $m = 1$ *to* M:

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ $\quad for\ i = 1,\dots,n$

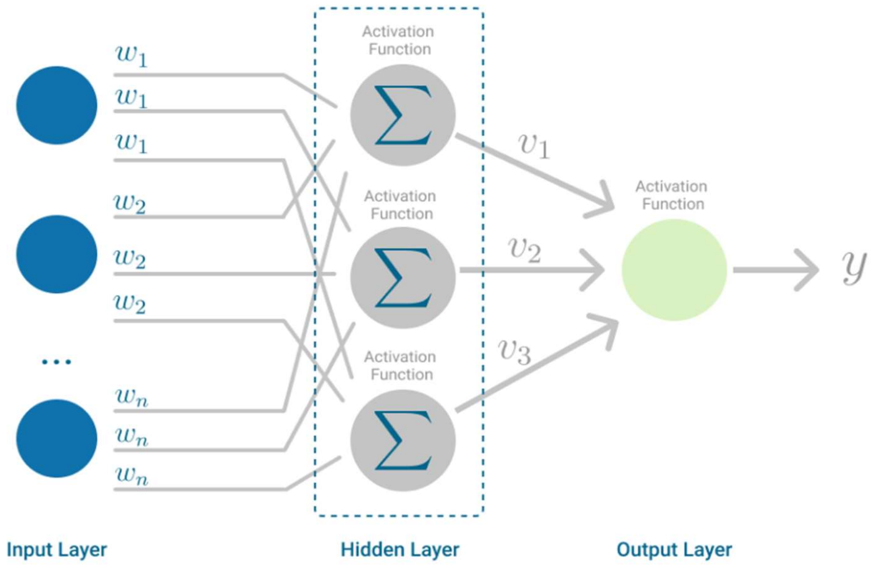
2-2. Train regression tree with features x against r and create terminal node reasions R_{jm} for $j = 1,\dots,J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ $\quad for\ j = 1,\dots,J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

Multilayer perceptron



Multilayer Perceptron. (Image by author)

4.3 Module Development –Code

1. **Logistic Regression**

```
# Linear regression model
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
```

```
log = LogisticRegression()

# fit the model
log.fit(X_train,y_train)
#predicting the target value from the model for the samples

y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print("Logistic Regression : Accuracy on training Data:
{:.3f}".format(acc_train_log))
print("Logistic Regression : Accuracy on test Data: {:.3f}".format(acc_test_log))
print()

f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print("Logistic Regression : f1_score on training Data:
{:.3f}".format(f1_score_train_log))
print("Logistic Regression : f1_score on test Data:
{:.3f}".format(f1_score_test_log))
print()

recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print("Logistic Regression : Recall on training Data:
{:.3f}".format(recall_score_train_log))
print("Logistic Regression : Recall on test Data:
{:.3f}".format(recall_score_test_log))
print()

precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print("Logistic Regression : precision on training Data:
{:.3f}".format(precision_score_train_log))
print("Logistic Regression : precision on test Data:
{:.3f}".format(precision_score_test_log))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_log))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Logistic Regression',acc_test_log,f1_score_test_log,
             recall_score_train_log,precision_score_train_log)
```

2. **k-Nearest Neighbors**

```
# 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)
#predicting the target value from the model for the samples
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
#computing the accuracy,f1_score,Recall,precision of the model performance

acc_train_knn = metrics.accuracy_score(y_train,y_train_knn)
acc_test_knn = metrics.accuracy_score(y_test,y_test_knn)
print("K-Nearest Neighbors : Accuracy on training Data:
{:.3f}".format(acc_train_knn))
print("K-Nearest Neighbors : Accuracy on test Data: {:.3f}".format(acc_test_knn))
print()

f1_score_train_knn = metrics.f1_score(y_train,y_train_knn)
f1_score_test_knn = metrics.f1_score(y_test,y_test_knn)
print("K-Nearest Neighbors : f1_score on training Data:
{:.3f}".format(f1_score_train_knn))
print("K-Nearest Neighbors : f1_score on test Data:
{:.3f}".format(f1_score_test_knn))
print()

recall_score_train_knn = metrics.recall_score(y_train,y_train_knn)
recall_score_test_knn = metrics.recall_score(y_test,y_test_knn)
print("K-Nearest Neighborsn : Recall on training Data:
{:.3f}".format(recall_score_train_knn))
print("Logistic Regression : Recall on test Data:
{:.3f}".format(recall_score_test_knn))
print()

precision_score_train_knn = metrics.precision_score(y_train,y_train_knn)
precision_score_test_knn = metrics.precision_score(y_test,y_test_knn)
print("K-Nearest Neighbors : precision on training Data:
{:.3f}".format(precision_score_train_knn))
print("K-Nearest Neighbors : precision on test Data:
{:.3f}".format(precision_score_test_knn))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_knn))
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
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();
#storing the results. The below mentioned order of parameter passing is important.

storeResults('K-Nearest Neighbors',acc_test_knn,f1_score_test_knn,
             recall_score_train_knn,precision_score_train_knn)
```

3. **Support Vector Classifier**

```
# Support Vector Classifier model
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'gamma': [0.1],'kernel': ['rbf','linear']}

svc = GridSearchCV(SVC(), param_grid)

# fitting the model for grid search
svc.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_train_svc = svc.predict(X_train)
y_test_svc = svc.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data:
{:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data: {:.3f}".format(acc_test_svc))
print()

f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data:
{:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data:
{:.3f}".format(f1_score_test_svc))
print()

recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data:
{:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data:
{:.3f}".format(recall_score_test_svc))
print()

precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
```

```
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data:
{:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data:
{:.3f}".format(precision_score_test_svc))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_svc))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Support Vector Machine',acc_test_svc,f1_score_test_svc,
             recall_score_train_svc,precision_score_train_svc)
```

4. **Naive Bayes**

```
# Naive Bayes Classifier Model
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline

# instantiate the model
nb=  GaussianNB()

# fit the model
nb.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_nb = nb.predict(X_train)
y_test_nb = nb.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_nb = metrics.accuracy_score(y_train,y_train_nb)
acc_test_nb = metrics.accuracy_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_nb))
print("Naive Bayes Classifier : Accuracy on test Data: {:.3f}".format(acc_test_nb))
print()

f1_score_train_nb = metrics.f1_score(y_train,y_train_nb)
f1_score_test_nb = metrics.f1_score(y_test,y_test_nb)
print("Naive Bayes Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_nb))
print("Naive Bayes Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_nb))
print()

recall_score_train_nb = metrics.recall_score(y_train,y_train_nb)
recall_score_test_nb = metrics.recall_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_nb))
print("Naive Bayes Classifier : Recall on test Data:
{:.3f}".format(recall_score_test_nb))
print()

precision_score_train_nb = metrics.precision_score(y_train,y_train_nb)
precision_score_test_nb = metrics.precision_score(y_test,y_test_nb)
```

```
print("Naive Bayes Classifier : precision on training Data:
{:.3f}".format(precision_score_train_nb))
print("Naive Bayes Classifier : precision on test Data:
{:.3f}".format(precision_score_test_nb))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_svc))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Naive Bayes Classifier',acc_test_nb,f1_score_test_nb,
             recall_score_train_nb,precision_score_train_nb)
```

5. Decision Tree

```
# 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)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.accuracy_score(y_test,y_test_tree)
print("Decision Tree : Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree : Accuracy on test Data: {:.3f}".format(acc_test_tree))
print()

f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print("Decision Tree : f1_score on training Data:
{:.3f}".format(f1_score_train_tree))
print("Decision Tree : f1_score on test Data: {:.3f}".format(f1_score_test_tree))
print()

recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print("Decision Tree : Recall on training Data:
{:.3f}".format(recall_score_train_tree))
print("Decision Tree : Recall on test Data: {:.3f}".format(recall_score_test_tree))
print()

precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print("Decision Tree : precision on training Data:
{:.3f}".format(precision_score_train_tree))
print("Decision Tree : precision on test Data:
{:.3f}".format(precision_score_test_tree))
```

```
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_tree))
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 30
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();
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Decision Tree',acc_test_tree,f1_score_test_tree,
             recall_score_train_tree,precision_score_train_tree)
```

6. **Random Forest**

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier

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

# fit the model
forest.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_forest = forest.predict(X_train)
y_test_forest = forest.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_forest = metrics.accuracy_score(y_train,y_train_forest)
acc_test_forest = metrics.accuracy_score(y_test,y_test_forest)
print("Random Forest : Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random Forest : Accuracy on test Data: {:.3f}".format(acc_test_forest))
print()

f1_score_train_forest = metrics.f1_score(y_train,y_train_forest)
f1_score_test_forest = metrics.f1_score(y_test,y_test_forest)
print("Random Forest : f1_score on training Data:
{:.3f}".format(f1_score_train_forest))
print("Random Forest : f1_score on test Data: {:.3f}".format(f1_score_test_forest))
print()

recall_score_train_forest = metrics.recall_score(y_train,y_train_forest)
recall_score_test_forest = metrics.recall_score(y_test,y_test_forest)
print("Random Forest : Recall on training Data:
{:.3f}".format(recall_score_train_forest))
print("Random Forest : Recall on test Data: {:.3f}".format(recall_score_test_forest))
print()

precision_score_train_forest = metrics.precision_score(y_train,y_train_forest)
precision_score_test_forest = metrics.precision_score(y_test,y_test_tree)
print("Random Forest : precision on training Data:
{:.3f}".format(precision_score_train_forest))
print("Random Forest : precision on test Data:
{:.3f}".format(precision_score_test_forest))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_forest))
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
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();
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Random Forest',acc_test_forest,f1_score_test_forest,
             recall_score_train_forest,precision_score_train_forest)
```

7. Gradient Boosting

```
# Gradient Boosting Classifier Model
from sklearn.ensemble import GradientBoostingClassifier

# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

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

```
#predicting the target value from the model for the samples
y_train_gbc = gbc.predict(X_train)
y_test_gbc = gbc.predict(X_test)
acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_gbc))
print("Gradient Boosting Classifier : Accuracy on test Data:
{:.3f}".format(acc_test_gbc))
print()

f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_gbc))
print("Gradient Boosting Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_gbc))
print()

recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc =  metrics.recall_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_gbc))
print("Gradient Boosting Classifier : Recall on test Data:
{:.3f}".format(recall_score_test_gbc))
print()

precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)
precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : precision on training Data:
{:.3f}".format(precision_score_train_gbc))
print("Gradient Boosting Classifier : precision on test Data:
{:.3f}".format(precision_score_test_gbc))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_gbc))
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  GradientBoostingClassifier(learning_rate = n*0.1)

    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 50
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("learning_rate")
plt.legend();
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10,1)
for n in depth:
    forest_test =  GradientBoostingClassifier(max_depth=n,learning_rate = 0.7)

    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 50
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("max_depth")
plt.legend();
```

8. Catboost

```
#  catboost Classifier Model
from catboost import CatBoostClassifier

# instantiate the model
cat = CatBoostClassifier(learning_rate  = 0.1)

# fit the model
cat.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_cat = cat.predict(X_train)
y_test_cat = cat.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_cat  = metrics.accuracy_score(y_train,y_train_cat)
acc_test_cat = metrics.accuracy_score(y_test,y_test_cat)
print("CatBoost Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_cat))
print("CatBoost Classifier : Accuracy on test Data: {:.3f}".format(acc_test_cat))
print()

f1_score_train_cat = metrics.f1_score(y_train,y_train_cat)
f1_score_test_cat = metrics.f1_score(y_test,y_test_cat)
print("CatBoost Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_cat))
print("CatBoost Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_cat))
```

```
print()

recall_score_train_cat = metrics.recall_score(y_train,y_train_cat)
recall_score_test_cat = metrics.recall_score(y_test,y_test_cat)
print("CatBoost Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_cat))
print("CatBoost Classifier : Recall on test Data:
{:.3f}".format(recall_score_test_cat))
print()

precision_score_train_cat = metrics.precision_score(y_train,y_train_cat)
precision_score_test_cat = metrics.precision_score(y_test,y_test_cat)
print("CatBoost Classifier : precision on training Data:
{:.3f}".format(precision_score_train_cat))
print("CatBoost Classifier : precision on test Data:
{:.3f}".format(precision_score_test_cat))
#computing the classification report of the model

print(metrics.classification_report(y_test, y_test_cat))
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  CatBoostClassifier(learning_rate = n*0.1)

    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 50
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("learning_rate")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important.

storeResults('CatBoost Classifier',acc_test_cat,f1_score_test_cat,
             recall_score_train_cat,precision_score_train_cat)
```

9. Xgboost

```
#  XGBoost Classifier Model
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier()

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

```
#predicting the target value from the model for the samples
y_train_xgb = xgb.predict(X_train)
y_test_xgb = xgb.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_xgb = metrics.accuracy_score(y_train,y_train_xgb)
acc_test_xgb = metrics.accuracy_score(y_test,y_test_xgb)
print("XGBoost Classifier : Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost Classifier : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
print()

f1_score_train_xgb = metrics.f1_score(y_train,y_train_xgb)
f1_score_test_xgb = metrics.f1_score(y_test,y_test_xgb)
print("XGBoost Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_xgb))
print("XGBoost Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_xgb))
print()

recall_score_train_xgb = metrics.recall_score(y_train,y_train_xgb)
recall_score_test_xgb = metrics.recall_score(y_test,y_test_xgb)
print("XGBoost Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_xgb))
print("XGBoost Classifier : Recall on test Data:
{:.3f}".format(recall_score_train_xgb))
print()

precision_score_train_xgb = metrics.precision_score(y_train,y_train_xgb)
precision_score_test_xgb = metrics.precision_score(y_test,y_test_xgb)
print("XGBoost Classifier : precision on training Data:
{:.3f}".format(precision_score_train_xgb))
print("XGBoost Classifier : precision on test Data:
{:.3f}".format(precision_score_train_xgb))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('XGBoost Classifier',acc_test_xgb,f1_score_test_xgb,
             recall_score_train_xgb,precision_score_train_xgb)
```

10. Multilayer Perceptrons

```
# Multi-layer Perceptron Classifier Model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier()
#mlp = GridSearchCV(mlpc, parameter_space)

# fit the model
mlp.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_mlp = mlp.predict(X_train)
y_test_mlp = mlp.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_mlp  = metrics.accuracy_score(y_train,y_train_mlp)
```

```
acc_test_mlp = metrics.accuracy_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Accuracy on training Data:
{:.3f}".format(acc_train_mlp))
print("Multi-layer Perceptron : Accuracy on test Data: {:.3f}".format(acc_test_mlp))
print()

f1_score_train_mlp = metrics.f1_score(y_train,y_train_mlp)
f1_score_test_mlp = metrics.f1_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : f1_score on training Data:
{:.3f}".format(f1_score_train_mlp))
print("Multi-layer Perceptron : f1_score on test Data:
{:.3f}".format(f1_score_train_mlp))
print()

recall_score_train_mlp = metrics.recall_score(y_train,y_train_mlp)
recall_score_test_mlp = metrics.recall_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Recall on training Data:
{:.3f}".format(recall_score_train_mlp))
print("Multi-layer Perceptron : Recall on test Data:
{:.3f}".format(recall_score_test_mlp))
print()

precision_score_train_mlp = metrics.precision_score(y_train,y_train_mlp)
precision_score_test_mlp = metrics.precision_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : precision on training Data:
{:.3f}".format(precision_score_train_mlp))
print("Multi-layer Perceptron : precision on test Data:
{:.3f}".format(precision_score_test_mlp))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Multi-layer Perceptron',acc_test_mlp,f1_score_test_mlp,
             recall_score_train_mlp,precision_score_train_mlp)
```

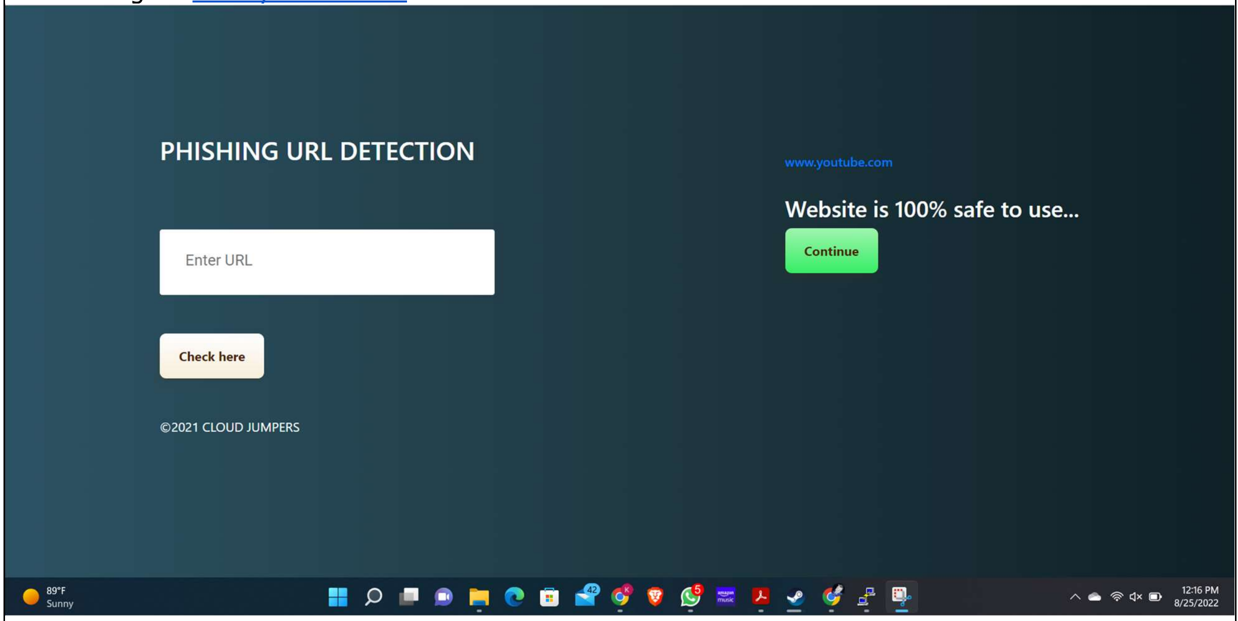
4.4 Test Cases

1. www.youtube.com
2. http://zqktlwiuavvvqqt4ybvgvi7tyo4hjl5xgfuvpdf6otjiycgwqbym2qad.onion/
3. https://tinyurl5.ru/e932441761/

5. Results and Discussion

5.1 Implementation Results

-> Checking for www.youtube.com



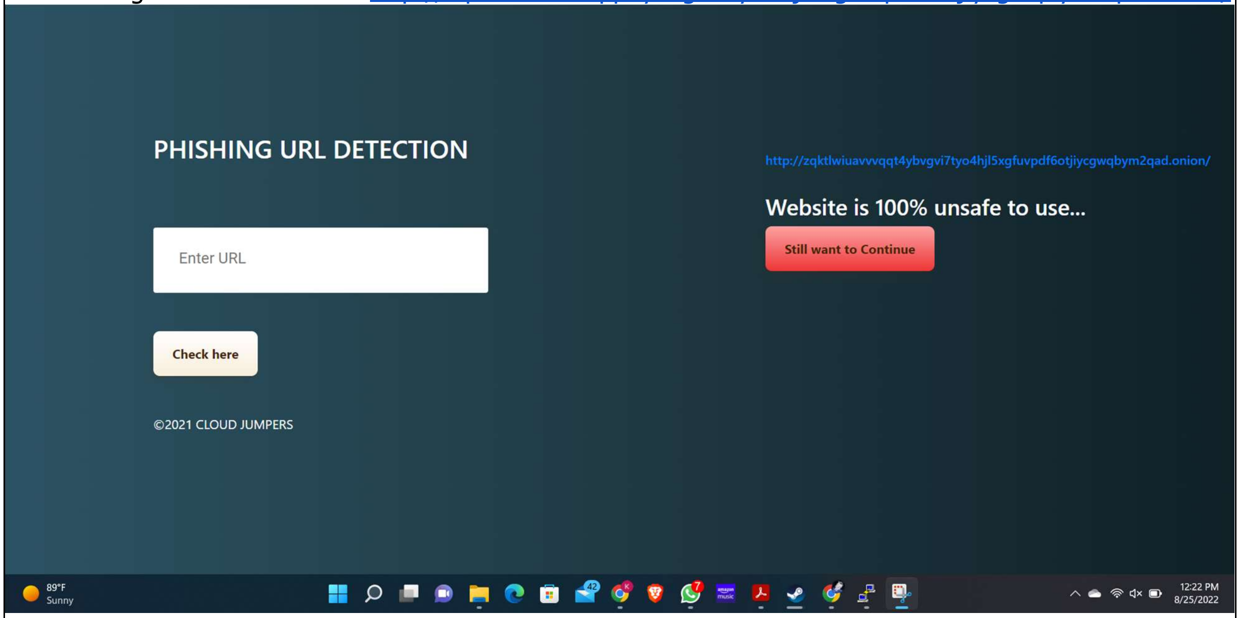
PHISHING URL DETECTION

©2021 CLOUD JUMPERS

www.youtube.com

Website is 100% safe to use...

-> Checking for an unsafe link: http://zqktlwiuavvvqqt4ybvgvi7tyo4hjl5xgfuvpdf6otjiycgwqbym2qad.onion/

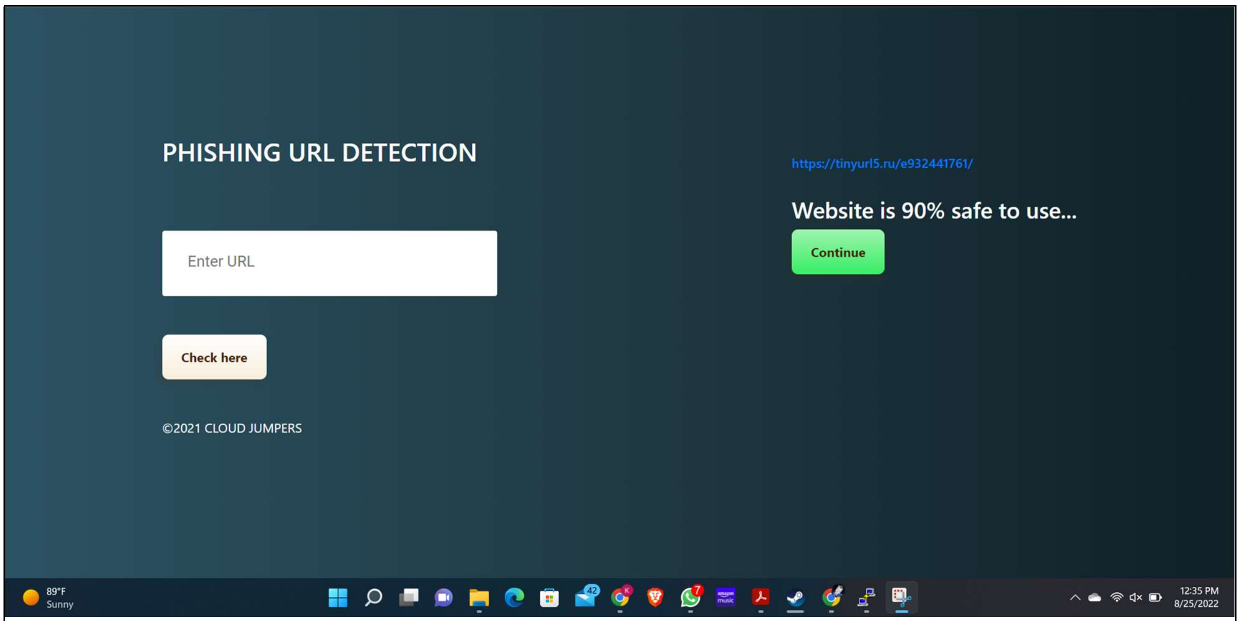


PHISHING URL DETECTION

©2021 CLOUD JUMPERS

http://zqktlwiuavvvqqt4ybvgvi7tyo4hjl5xgfuvpdf6otjiycgwqbym2qad.onion/

Website is 100% unsafe to use...


PHISHING URL DETECTION

Enter URL

Check here

©2021 CLOUD JUMPERS

https://tinyurl5.ru/e932441761/

Website is 90% safe to use...

Continue

5.2 Metrics

As seen from the above results, when using the online URL detection software it warns the users of the threat level of the website link they are trying to access. In the first image we have used the well known www.youtube.com, for which the result was 100% safe. In the second photo to demonstrate what kind of phishing links there are, we have used an onion link for which the result is 100% unsafe. This shows that the URL detection algorithm is able to identify the threat level of the entered website since onion links are very unsafe and pose a great threat to the user. To justify this, in the last photo we have used a link which we generally receive on our phone messages. Upon passing it through the algorithm we found out it is 90% safe, letting the user know it is not as safe as YouTube, but not as unsafe as an onion link. This gives the user an idea till to what extent the website links are safe/unsafe.

5.3 Results in table/Graph/Data (No screenshots, only text form of data in table), Graph should be drawn using Excel tool.

ML Model	**Accuracy**	**f1_score**	**Recall**	**Precision**
Logistic Regression	0.934	0.941	0.943	0.927
K-Nearest Neighbors	0.956	0.961	0.991	0.989
Support Vector Machine	0.964	0.968	0.980	0.965
Naive Bayes Classifier	0.605	0.454	0.292	0.997
Decision Tree	0.960	0.964	0.991	0.993
Random Forest	0.967	0.971	0.993	0.990
Gradient Boosting Classifier	0.974	0.977	0.994	0.986

CatBoost Classifier	0.972	0.975	0.994	0.989
XGBoost Classifier	0.969	0.973	0.993	0.984
Multi-Layer Perceptron	0.969	0.973	0.995	0.981

5.4 Mapping the results with problem statement and existing systems

The result of this project aims at identifying the phishing links, but as mentioned earlier the website can pose some level of threat and does not have to be either 100% safe/unsafe. The classifying systems existing in the market generally offer binary classification, that means a website which is 52% safe will be considered as completely safe, instead of a phishing link. To overcome this we have provided a percentage display of the website link to let the user till what the website is safe. In the modern world attackers are evolving at a fast rate and to avoid the phishing links which aim to capture your personal data, we recommend a URL detection software to check whether the entered website is safe or not and if it is then to what extent.

5.5 Discussions

As mentioned earlier that attackers are evolving at an alarming rate, that means so are the phishing links. The project developed has proved to be useful in classifying the links in safe/unsafe, based on the dataset used. But time and again there will be a necessity to train and evaluate the model again with new dataset so as to incorporate the new generation phishing links. By doing this we ensure that the algorithm used stays at par with the world of attackers, and that no link is missed.

6. Conclusion and Future Developments

The final take away from this project is to explore various machine learning models, perform Exploratory Data Analysis on phishing dataset and understand their features.
Creating this notebook helped me to learn a lot about the features affecting the models to detect whether URL is safe or not, also I came to know how to tune models and how they affect the model performance.
The final conclusion on the Phishing dataset is that the same features like "HTTPS", "AnchorURL", "WebsiteTraffic" have more importance to classify whether a URL is a phishing URL or not.
Gradient Boosting Classifier correctly classifies URL upto 97.4% respective classes and hence reduces the chance of malicious attachments.

7. References

1. Singh C, Meenu. Phishing Website Detection Based on Machine Learning: A Survey. 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). 2020

2. Yu WD, Nargundkar S, Tiruthani N. PhishCatch - A Phishing Detection Tool. 2009 33rd Annual IEEE International Computer Software and Applications Conference. 2009;

3. Shreeram V, Suban M, Shanthi P, Manjula K. Anti-phishing detection of phishing attacks using genetic algorithm. 2010 INTERNATIONAL CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING TECHNOLOGIES. 2010

4. Chen J, Guo C. Online Detection and Prevention of Phishing Attacks. 2006 First International Conference on Communications and Networking in China. 2006

5. Lew May Form, Kang Leng Chiew, San Nah Sze, Wei King Tiong. Phishing email detection technique by using hybrid features. 2015 9th International Conference on IT in Asia (CITA). 2015

6. Niu W, Zhang X, Yang G, Ma Z, Zhuo Z. Phishing Emails Detection Using CS-SVM. 2017 IEEE

International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). 2017

7. Kiruthiga R, Akila D. Phishing Websites Detection using Machine Learning. International Journal of Recent Technology and Engineering. 2019;8(2):111-4.

8. Jain AK, Gupta BB. A machine learning based approach for phishing detection using hyperlinks information. Journal of Ambient Intelligence and Humanized Computing. 2018;10(5):2015-28.

9. Yang P, Zhao G, Zeng P. Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning. IEEE Access. 2019;7:15196-209.

10. Opara C, Wei B, Chen Y. HTMLPhish: Enabling Phishing Web Page Detection by Applying Deep Learning Techniques on HTML Analysis. 2020 International Joint Conference on Neural Networks (IJCNN). 2022

11. Do, N. Q., Selamat, A., Krejcar, O., Herrera-Viedma, E., & Fujita, H. (2022). Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions. *IEEE Access*.

12. Atre, M., Jha, B., & Rao, A. (2022). Detecting Cloud-Based Phishing Attacks by Combining Deep Learning Models. *arXiv preprint arXiv:2204.02446*.

13. Alqahtani, H., Alotaibi, S. S., Alrayes, F. S., Al-Turaiki, I., Alissa, K. A., Aziz, A. S. A., ... & Al Duhayyim, M. (2022). Evolutionary Algorithm with Deep Auto Encoder Network Based Website Phishing Detection and Classification. *Applied Sciences*, *12*(15), 7441.

14. Mohamed, G., Visumathi, J., Mahdal, M., Anand, J., & Elangovan, M. (2022). An Effective and Secure Mechanism for Phishing Attacks Using a Machine Learning Approach. *Processes*, *10*(7), 1356.

15. Kumar, C. S., & Iyakutti, K. (2014). A novel technique: Data leakage hindering in cloud computing using swarm intelligence. *Int. J. Technol. Appl*, *5*(6), 1886-1891.

16. Chhabra, S., & Singh, A. K. (2016, December). Dynamic data leakage detection model based approach for MapReduce computational security in cloud. In *2016 Fifth International Conference on Eco-friendly Computing and Communication Systems (ICECCS)* (pp. 13-19). IEEE.

17. Tihanyi, V., Rövid, A., Remeli, V., Vincze, Z., Csonthó, M., Pethő, Z., ... & Szalay, Z. (2021). Towards Cooperative Perception Services for ITS: Digital Twin in the Automotive Edge Cloud. *Energies*, *14*(18), 5930.

18. Cao, C., Blaise, A., Verwer, S., & Rebecchi, F. (2022). Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster. *arXiv preprint arXiv:2207.12087*.

19. Motahari-Nezhad, H., Stephenson, B., & Singhal, S. (2009). Outsourcing business to cloud computing services. *Hewlett-Packard Technical Report HPL-2009-23. Available at: http://www. hpl. hp. com/techreports/2009/HPL-2009-23. pdf*.

20. Kumar, C. S., & Iyakutti, K. (2014). A novel technique: Data leakage hindering in cloud computing using swarm intelligence. *Int. J. Technol. Appl*, *5*(6), 1886-1891.

21. Li, T., Han, F., Ding, S., & Chen, Z. (2011, July). Larx: Large-scale anti-phishing by retrospective data-exploring based on a cloud computing platform. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)* (pp. 1-5). IEEE

22. Chen, Z., Han, F., Cao, J., Jiang, X., & Chen, S. (2013). Cloud computing-based forensic analysis for collaborative network security management system. *Tsinghua science and technology*, *18*(1), 40-50.

23. Mourtaji, Y., Bouhorma, M., Alghazzawi, D., Aldabbagh, G., & Alghamdi, A. (2021). Hybrid rule-based solution for phishing URL detection using convolutional neural network. *Wireless Communications and*

Mobile Computing, *2021*.

24. Ferguson, E., Weber, J., & Hasan, R. (2012, June). Cloud based content fetching: Using cloud infrastructure to obfuscate phishing scam analysis. In *2012 IEEE Eighth World Congress on Services* (pp. 255-261). IEEE.

25. Canova, G., Volkamer, M., Bergmann, C., Borza, R., Reinheimer, B., Stockhardt, S., & Tenberg, R. (2015, May). Learn to spot phishing URLs with the android NoPhish app. In *IFIP World Conference on Information Security Education* (pp. 87-100). Springer, Cham.

26. Abdullayeva, F. J. (2021). Advanced Persistent Threat attack detection method in cloud computing based on autoencoder and softmax regression algorithm. *Array*, *10*, 100067.

27. Lin, C. H., Tien, C. W., Chen, C. W., Tien, C. W., & Pao, H. K. (2015, September). Efficient spear-phishing threat detection using hypervisor monitor. In *2015 International Carnahan Conference on Security Technology (ICCST)* (pp. 299-303). IEEE.

28. Rao, P. M., & Saraswathi, P. (2021). Evolving cloud security technologies for social networks. In *Security in IoT Social Networks* (pp. 179-203). Academic Press.

29. Singh, J. (2014). Cyber-attacks in cloud computing: A case study. *International Journal of Electronics and Information Engineering*, *1*(2), 78-87.

30. Butt, U. A., Amin, R., Aldabbas, H., Mohan, S., Alouffi, B., & Ahmadian, A. (2022). Cloud-based email phishing attack using machine and deep learning algorithm. *Complex & Intelligent Systems*, 1-28.

Research article must be from reputed journal with TR- Impact Factor only

Documentation without Plagiarism is most important, Turnitin software will be used to evaluate. (Making false sentences, paraphrasing, changing characters in the document, inserting images as paragraph for the purpose of reducing plagiarism will lead to marking ZERO).

Google Colab "Phishing URL Detection" code :
https://colab.research.google.com/drive/1MJi3JAS3ZkuE6BZr5Uz93uomHvFmRzxG?usp=sharing

VIT® Vellore Institute of Technology (Deemed to be University under section 3 of UGC Act, 1956)	**B.Tech (Information Technology)** **SUMMER SEMESTER TERM 5 - 2022** **ITE 3007 : Cloud Computing and Virtualization** **19.08.2022**

Title: Phishing Link Detection using Cloud Computing

Team Name: Team Jumpers

Project Team

S.No	**Register Number**	**Student Name**	**Signature**	**Guided By**
1	19BIT0256	Kunj Patel	Kunj Patel	**Dr. Nadesh R.K**
2	19BIT0275	Tanishq	[illegible]	
3	19BIT0266	Yash Sen	Yash	

Team Member(s) Contribution and Performance Assessment

Components	**Student 1**	**Student 2**	**Student 3**
Implementation & Results -(20)			
Contributed fair share to the team project -(05)			
Documentation without Plagiarism - (20)			
Q & A - (05)			

Student Feedback (Student Experience in this Course Project)	**Evaluator Comments**

Name & Signature of the Evaluator

(Dr.Nadesh R.K)