International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Cloud-Based Machine Learning Approach for Accurate Detection of Website Phishing

Saba Hussein Rashid*[1], Wisam Dawood Abdullah[2]

Submitted: 10/02/2023 **Revised:** 13/04/2023 **Accepted:** 12/05/2023

Abstract: In recent years, the rapid growth of cyberspace has led to an increase in challenges related to information security. One of the most dangerous cyberattacks is website phishing, which is complex in nature and difficult to detect in real-time. Cloud Machine Learning has emerged as an effective approach for detecting website phishing by leveraging Cloud Computing Services to obtain accurate results quickly. Therefore, this study presents a Cloud Machine Learning method for evaluating and assessing the time required to detect website phishing using three SageMaker built-in algorithms: Extreme Gradient Boosting, Linear Learner, and k-Nearest Neighbor. Amazon Web Services is utilized for storage, training, evaluation, and online deployment over a large dataset of 11,430 samples and 89 features. The results indicate that Extreme Gradient Boosting outperformed the other two algorithms with an accuracy of 96.4% and an online prediction time of 0.0005 minutes, followed by Linear Learner with an accuracy of 94.4% and a prediction time of 0.0006 minutes. While k-Nearest Neighbor obtained the lowest accuracy score of 83.7% and the longest prediction time of 0.0008 minutes.

Keywords: *AWS, Cloud Machine Learning, Prediction Time, Website Phishing, XGBoost*

1. Introduction

Cybersecurity refers to the tools, actions, and technologies employed to safeguard cyberspace and its assets, emphasizing confidentiality, availability, and integrity [1]. The increasing usage of the Internet and information systems in nearly every aspect of daily life in cyberspace has led to a corresponding surge in cyberattacks against users and organizations. To combat these attacks, new methods need to be created, as cybercriminals continue to develop novel techniques to spread malicious entities [2]. One of the most commonly used social engineering attacks is website phishing, which involves creating an identical copy or alternating a legitimate webpage to gather confidential information from users who are unable to differentiate between fake and legitimate sites. These attacks are typically aimed at financial, online shopping, and payment services, with the primary goal of obtaining financial gain from the victims [3].

Detecting website phishing is a complex task that requires processing large amounts of data in real-time. Significant progress has been made in recent years regarding the detection of phishing attacks through the use of various Machine Learning systems. However, much work still needs to be done to enhance the scalability, speed, and accuracy of existing systems. Different studies have proposed various Machine Learning systems that leverage algorithms like Extreme Gradient Boosting (XGBoost) and k-Nearest Neighbor (k-NN) to identify phishing attacks.[4]–[14], which could achieve accurate results. However, they have encountered difficulties concerning storage capacity and computing resources. The ability to identify phishing attacks in real-time is vital to prevent possible harm to users. However, many existing systems face challenges in achieving this objective due to the inadequacy of machines in providing the necessary resources to analyze data promptly without depending on third-party resources.

Cloud Machine Learning is an emerging field that leverages Cloud Computing Services to construct, train, and deploy effective models capable of obtaining rapid and precise results, while also managing big data through the use of collaborative computing interfaces and powerful virtual resources like storage, databases, graphics, networks, and processing. Compared to traditional on-premise Machine Learning, Cloud Machine Learning offers numerous benefits such as scalability, cost-effectiveness, and ease of use [15]. Amazon Web Services (AWS) is a Cloud Computing platform owned by Amazon and is offered on a pay-as-you-go basis to cater to various users and organizations. It provides developers with the ability to control their virtual machines without the need to worry about the underlying structure of these services. AWS offers a comprehensive suite of hardware and software resources, including virtual CPUs, GPUs, storage, servers, and

[1]*College of Computer Science and Mathematics, Tikrit University, Salaheddin, IRAQ*
ORCID ID : 0009-0005-8239-7279
[2] *Asst. Prof., College of Computer Science and Mathematics, Tikrit University, Salaheddin, IRAQ*
ORCID ID : 0000-0001-9517-5994
** Corresponding Author Email: sabahussein88@tu.edu.iq*

operating systems, as well as pre-installed applications. Among the key services provided by AWS are SageMaker, Simple Storage Service (S3), CloudWatch, and Elastic Compute Cloud (EC2) [16], [17].

This study introduces a Cloud Machine Learning approach that employs Amazon Web Services (AWS) to evaluate the time required for training and predicting website phishing attack detection. The proposed method involves the use of three algorithms, Extreme Gradient Boosting (XGBoost), Linear Learner, and k-Nearest Neighbor (k-NN), and measures the processing time for training, batch transform, and online prediction on a large dataset of 11,430 samples that have been pre-processed and stored on Amazon Simple Storage Service (S3). The model is built, trained, evaluated, and deployed using Amazon SageMaker, while performance monitoring is conducted through Amazon CloudWatch. The virtual resources required for model-building are obtained from Amazon Elastic Compute Cloud (EC2) instances.

2. Methodology

The proposed method for phishing attack detection involved four main stages as shown in Fig. 1. These stages were data collection, data preprocessing, training and evaluating the proposed method using SageMaker built-in algorithms, and online deployment. In the first stage, the dataset was downloaded from Mendeley Data, visualized to understand the behavior of its features, and uploaded to Amazon Simple Storage Service (S3) bucket. The second stage involved performing five preprocessing steps to obtain the most important features, including feature selection, label encoding, standardization, feature scaling, and dataset splitting.

The third stage utilized SageMaker built-in algorithms, Extreme Gradient Boosting (XGBoost), Linear Learner, and k-Nearest Neighbor (k-NN), to train and evaluate the proposed method, with SageMaker Batch Transform used for performance evaluation. Five metrics were used for displaying the evaluation results: accuracy, precision, F-1, recall, and mislabeling. In the final stage, the proposed method was deployed using a multi-model endpoint in SageMaker Inference Endpoints, enabling real-time prediction.

Amazon CloudWatch is used to monitor performance and processing time. Computational resources were sourced from Amazon Elastic Compute Cloud (EC2), specifically the (ml.m4.xlarge) instance that provided 4 virtual Central Processing Units (CPUs), 16 GB of Random Access Memory (RAM), and high network performance for both 32 and 64-bit virtual operating systems.

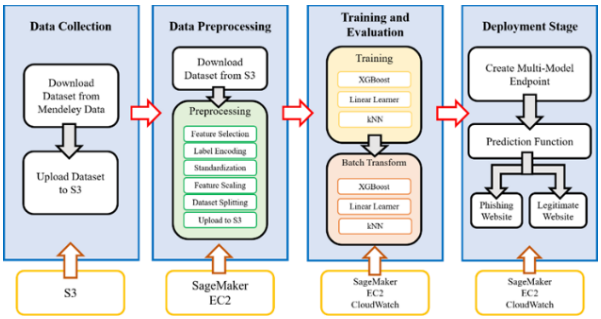

Fig. 1. The implementation stages of the research methodology

2.1. Dataset Collection and Description

This paper employed the "Web Page Phishing Detection" dataset from Mendeley Data to develop and test the proposed method. This dataset was preferred because of its ample size and detailed features, enabling a better understanding of website properties. As shown in Table 1, the dataset was a binary classification dataset that included 11,430 Uniform Resource Locators (URLs) and 89 features, consisting of 87 numeric, one nominal feature representing the name of the URL, and one categorical feature representing the labeling of each website. The dataset was balanced, with exactly 50% (5715) of its samples labeled as "legitimate" and 50% (5715) as "phishing" as shown in Fig. 2, and it was devoid of missing values as shown in Fig. 3. After downloading it from the source, the dataset was uploaded and stored in Amazon Simple Storage Service (AWS S3) bucket.

Table 1. Description of the dataset

Characteristic	*Value*
Name	**Web Page Phishing Detection**
Source	**Mendeley Data**
Type	**Binary/Supervised**
Number of Samples	**11,430**
Number of Extracted Features	**87**

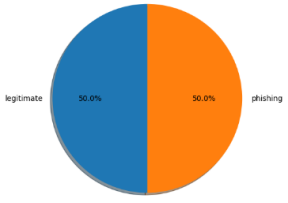

Fig. 2. The ratio of balancing of the labels of the binary dataset

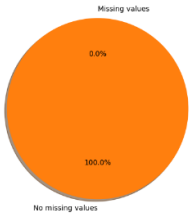

Fig. 3. The ratio of missing values in the dataset

2.2. Dataset Analysis

In this study, a comprehensive analysis was conducted to understand the behavior of the dataset to achieve better accuracy results. The analysis categorized the 87 extracted features into three groups: (1) 56 features derived from the syntax of the Uniform Resource Locators (URL), (2) 24 features derived from the content of the websites, and (3) 7 features derived from the services associated with the websites. Upon examining some of the dataset's features, it was observed that most websites had a URL length between 1 and 200 characters, as depicted in Fig. 4. Furthermore, the length of hostnames varied across the URLs in the dataset, with 10,000 of the URLs having a hostname length between 1 and 25, as shown in Fig. 5. The dataset analysis revealed several key findings about the features. One of the significant findings is that out of the 11,430 URLs in the dataset, 53.4% of them (6103 URLs) had a Google Index ranking, while the remaining 46.6% (5327 URLs) did not as illustrated in Fig. 6. This is a crucial factor in assessing the credibility of the website. Another observation is that 84.9% of the URLs (9709 URLs) had IP addresses associated with their websites, while 15.1% of them (1721 URLs) did not. However, the absence of an IP address could not necessarily determine the legitimacy of a website, these findings are depicted in Fig. 7.

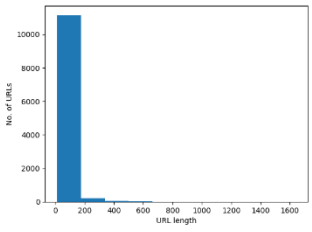

Fig. 4. The length of the URLs in the dataset

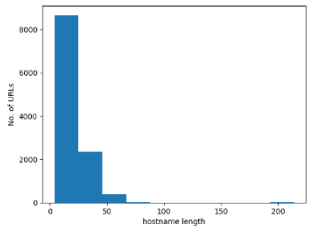

Fig. 5. The length of hostnames of the URLs in the dataset

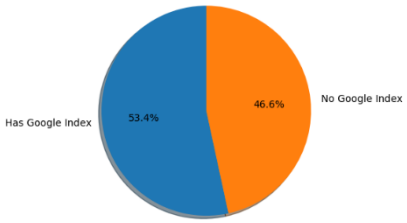

Fig. 6. The ratio of URLs with Google Index in the dataset

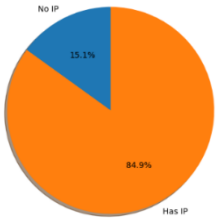

Fig. 7. The ratio of URLs with IP addresses in the dataset

2.3. Dataset Preprocessing

Amazon SageMaker has the ability to handle large datasets efficiently, but a preprocessing stage is essential to obtain a fast and interpretable model. The preprocessing stage consists of five steps: feature selection, label encoding, standardization, feature scaling, and dataset splitting. These steps are crucial in simplifying the data, accelerating the training and evaluation processes, and avoiding overfitting. The procedures of the preprocessing step are presented in Fig. 8. It was downloaded into the SageMaker Studio Notebook to preprocess the dataset and read into a Python script file. The feature selection step used Pearson Correlation to remove highly correlated features. The label encoding function was applied to convert categorical values of the target feature to numeric values. The standardization step employed the StandardScaler() to prevent the dominance of large values over small values. Finally, the MinMaxScaler() was used in the scaling step to ensure that all features were on a similar scale between 0 and 1. Once the preprocessing stage was completed, the dataset was ready for splitting.

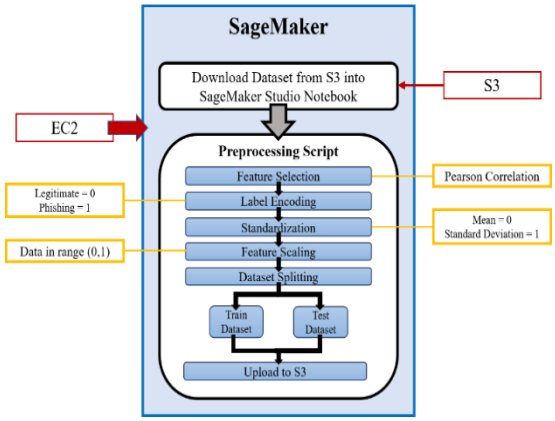

Fig. 8. Steps of the preprocessing stage

2.3.1. Feature Selection

The feature selection process is crucial in machine learning, as it involves identifying the most relevant features from the dataset to improve the model's accuracy. By removing irrelevant and redundant features, feature selection can reduce the dimensionality of the dataset, enhance model performance, and avoid overfitting. This study used Pearson Correlation to perform feature selection and extract important features. The Pearson correlation coefficient is a standard test used to assess the statistical relationship between two variables. The correlation matrix, a square matrix that displays the correlation coefficient between each pair of variables in the dataset, was used to visualize the interdependence and direction of the relationship between variables. The correlation matrix for this study revealed that 9 of the features correlated higher than 70%, indicating a high correlation among these features. This matrix could be visualized using a heatmap, where darker colors indicate a stronger correlation between variables.

2.3.2. Label Encoding

Label encoding is a process that transforms categorical data into numerical data, allowing algorithms to better process and analyze the data. The label encoding process assigns a unique numerical value to each unique category within a feature. For example, in a binary classification problem, the label encoding process would assign a numerical value to each label. In this study, the scikit-learn library LabelEncoder() was used to convert the categorical values of the target feature. This function assigned the numerical label of 0 to the categorical label "legitimate" and the numerical label of 1 to the categorical label "phishing."

2.3.3. Standardization

Standardization is a common preprocessing step in machine learning that rescales the features of a dataset to have a mean of zero and a standard deviation of one. This is done to prevent any feature from dominating the others. Standardization is particularly useful for algorithms that are sensitive to feature scales, such as support vector machines and k-nearest neighbors. In this study, the StandardScaler() function was used to standardize 12 features of the dataset that had very large values to ensure that all the features were on a similar scale.

2.3.4. Feature Scaling

Feature scaling is a data preprocessing technique used to normalize the range of independent variables or features in a dataset. It is often used in machine learning algorithms to ensure that the features are on a similar scale and have the same range. Normalization scales the data to fit within a specific range, usually between zero and one. In this study, the MinMaxScaler() function was used to scale the data between the range of zero and one, where the minimum value of the feature was mapped to zero, and its maximum value was mapped to one.

2.3.5. Dataset Splitting

Dataset splitting involves dividing a dataset into two parts: a training set and a test set. The training set is used to train the model, while the test set is used to evaluate its performance. The goal of dataset splitting is to ensure that the model can generalize well to new, unseen data. If the model is only trained on a single dataset, it may overfit to that particular dataset and not perform well on new data. By testing the model on a separate test set, we can get a more accurate estimate of its performance on new data. This study used the train_test_split() function from the scikit-learn library to split the dataset into training and test sets. The function split the dataset into two sets of 70:30 ratio, where 70% of the dataset (8001 samples) was used for training and the remaining 30% (3425 samples) was used for testing the model.

2.4. Building the Model

Amazon SageMaker is a fully-managed service that provides tools and libraries for building, training, and deploying Cloud Machine-Learning algorithms. SageMaker contains pre-built Docker containers that can be used to run many built-in algorithms and train the model at a high scale. In this study, three SageMaker built-in algorithms were used to train the model: Extreme Gradient Boosting (XGBoost), Linear Learner, and k-Nearest Neighbor (k-NN).

2.4.1. Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a Machine Learning algorithm designed for classification and regression, which is highly scalable and offers fast, accurate models for solving complex problems. XGBoost employs a Tree Structure model and Parallel Threading Computing to achieve high-speed performance, scalability, and portability, particularly for big data. The algorithm uses a sequential training approach, where each new decision tree is trained to correct the errors of the previous tree, enabling XGBoost to learn from its mistakes and improve over time. XGBoost includes a regularization term in the objective function to prevent overfitting and improve generalization performance, and by that, penalizes complex models and encourages simpler models that are less likely to overfit the training data. During training, XGBoost calculates the gradient of the objective function concerning the model parameters using backpropagation and uses it to update the model parameters and enhance its performance. In binary classification, given a training set with N observations, $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where x_i is the input vector and y_i is the binary output, we want to learn a model $F(x)$ that predicts the probability of the positive class given a new input vector x as shown in eq. (1).

$$F(x) = P(y = 1|x) \quad (1)$$

To learn the model *F(x)*, we need to minimize the following objective function in eq. (2):

$$obj = F_{obj} + \Omega(\mathrm{F}) \quad (2)$$

Where F_{obj} is the training objective function that measures the difference between the predicted probabilities and the true labels, and Ω(F) is the regularization term that penalizes the complex models as in eq. (3).

$$F_{obj} = \sum_{i=1}^{N} L(y_i, F(x_i)) + \sum_{j=1}^{K} 1\Omega(f_j) \quad (3)$$

Where L is the loss function that measures the difference between the predicted probabilities and the true labels, and f_j is the *j*-th weak prediction model. The objective function F_{obj} can be optimized using gradient boosting. The gradient boosting algorithm iteratively adds new weak prediction models to the ensemble, where each new model is trained to predict the residual errors of the previous models. The final prediction is the sum of the predictions of all the weak models as in eq. (4):

$$F(x) = \sum_{j=1}^{K} f_j(x) \quad (4)$$

Where f_j is the *j*-th weak prediction model. Each weak model is trained by minimizing the following objective function as eq. (5):

$$obj_j = \sum_{i=1}^{N} L\left(y_i, f_{j-1}(x_i)\right) + \Omega(f_j) \quad (5)$$

Where f_{j-1} is the ensemble of the previous (j-1) models, and f_j is the *j*-th weak model to be added to the ensemble. The objective function obj_j can be optimized using gradient boosting by fitting the negative gradient of obj_j as the new target as in eq. (6):

$$-\left(\partial obj_j \partial f_j(x_i)\right) = y_i - f_j - 1(x_i) \quad (6)$$

The negative gradient represents the residual errors that are not captured by the previous models. The new weak model f_j is then fitted to predict the negative gradient. This process is repeated until the training objective function F_{obj} converges. [18]–[22].

2.4.2. Linear Learner

Linear Learner, a machine learning algorithm built into SageMaker, is suitable for both classification and regression tasks. This algorithm is particularly useful for large datasets and implements Stochastic Gradient Descent to update the model weights iteratively. Linear Learner also supports automatic feature scaling to ensure that features with different scales can be used without one feature dominating the others. An essential feature of Linear Learner is its inclusion of a regularization term in the objective function, which prevents overfitting and improves generalization performance by favoring simpler models. Due to these features, Linear Learner is commonly employed in fraud detection, banking, and health management models. Moreover, Linear Learner can tune hyperparameters automatically using AutoML, which further improves its efficiency. The Linear Learner algorithm uses logistic regression to model the probability of a binary outcome. The logistic regression model can be written as in eq. (7):

$$P(y = 1|x, w, b) = \sigma(\mathrm{w}T_x + \mathrm{b}) \quad (7)$$

Where *y* is the binary outcome (1 or 0), *x* is the input vector, *w* is the weight vector, *b* is the bias term, and σ is the sigmoid function, which maps the input to the range [0,1].

The objective of the Linear Learner algorithm is to minimize the binary cross-entropy loss function, which is defined as in eq. (8):

$$L(w, b) = -\left[y \log_p(y = 1|x, w, b)\right] + (1 - y) \log_{1-p}(y = 1|x, w, b) \quad (8)$$

The loss function measures the difference between the predicted probability and the true label for each training example. The algorithm minimizes this loss function using stochastic gradient descent (SGD) with mini-batch updates. To minimize the cost function, linear learner uses an optimization algorithm called stochastic gradient descent (SGD), which updates the weights and bias in the direction of the negative gradient of the cost function. The update rule for SGD can be written mathematically as in eq. (9) and eq. (10):

$$w_i = w_i - learning_{rate} * d_j(w, b)/d_{w_i} \quad (9)$$

$$b = b - learning_{rate} * d_j(w, b)/d_b \quad (10)$$

Where $learning_{rate}$ is a hyperparameter that controls the step size of the updates, and $d_j(w, b)/d_{w_i}$ and $d_j(w, b)/d_b$ are the partial derivatives of the cost function with respect to the weights and bias, respectively. The above equations are iteratively updated until the predefined maximum number of iterations is reached. To prevent overfitting, the algorithm employs L1 and L2 regularization techniques. L1 regularization adds a penalty term to the loss function proportional to the absolute values of the weight vector, while L2 regularization adds a penalty term proportional to the square of the weight vector. The regularization parameter is controlled by a hyperparameter lambda, which determines the strength of regularization. [23].

2.4.3. k-Nearest Neighbor (k-NN)

k-Nearest Neighbor (k-NN) is a widely used Machine-Learning algorithm for classification problems and it follows the Lazy Learning method. It determines the class of data point by examining the majority of its neighbors, who share the same feature characteristics. Although the algorithm is known for its simplicity, it can be computationally expensive and suffer from the curse of dimensionality, particularly when working with large datasets. k-NN works by selecting the k-nearest data points to a query point based on a distance metric and then predicting the label of the query point based on the most common label among its k-nearest neighbors. The Euclidean distance is commonly used as a distance metric, but other metrics such as Manhattan distance and cosine similarity can be utilized. In this paper, the proposed technique is evaluated by utilizing the k-NN classifier to investigate classification performance. The appropriate *k* value selection is critical as it impacts classification performance, and it can be achieved by calculating the Euclidean distance function between the testing set *q* and all training sets *p* as in eq. (11):

$$\boldsymbol{d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}} \quad \textbf{(11)}$$

Where *n* represents the number of features. The distance is then organized in ascending order to categorize the features based on k data and classify the new data.

Classification is carried out by considering the k-nearest neighbors with the smallest distances, where *k* denotes the number of nearest neighbors involved in the majority voting process. The test sample's class label is determined based on the majority votes of its k-nearest neighbors, corresponding to the class with the highest number of members in *k*. This process is performed according to eq. (12):

$$C(x_i) = \arg max_k \sum_{x_j \in KNN} C(x_j, y_k) \quad \textbf{(12)}$$

Where x_i is a test object, x_j is one of its k-nearest neighbors in the training set, $C(x_j, y_k)$ indicates whether x_j belongs to class y_k. [24]–[27].

3. Results and Discussion

This method used batch transform after training to obtain the evaluation results for each algorithm, then created a multi-model Endpoint for online deployment. The times required for training, batch transform, and endpoint prediction were calculated to indicate the fastest method to obtain accurate prediction results.

3.1. Evaluation results

The evaluation of the model was conducted in two steps: Batch Transform and Endpoint prediction. Batch transform prediction is a technique for making predictions on large datasets by processing data in batches and storing the prediction artifacts in Amazon Simple Storage Service (S3) bucket. Endpoint prediction, on the other hand, is a technique for making predictions by deploying a model as a web service endpoint that can receive Hypertext Transfer Protocol (HTTP) requests and respond with predictions in real-time. The test dataset was applied to obtain the evaluation metrics and measure the prediction time for both techniques.

3.1.1. Batch Transform Results

Batch Transform is a SageMaker feature designed to streamline and scale the process of making predictions from a trained model on large datasets, without latency or online deployment concerns. This is achieved by dividing the workload into records or mini-batches of a specific size, upon which the predictions are performed. Fig. 9 illustrates the steps involved in the Batch Transform process. Once the training results are obtained from Amazon Simple Storage Service (S3), the test dataset is used to initiate Batch Transform on the three algorithms of the model. Batch Transform provides valuable insights into the progress and performance of the transform job and is ideal for obtaining prediction results for one-time processing and storing them in Amazon Simple Storage Service (S3). However, the setup process for batch transform prediction can be time-consuming since it requires creating and configuring several resources and batch transform jobs. As a result, it is not suitable for real-time predictions. Table 2 displays the evaluation metrics of three machine learning algorithms after Batch Transform. Extreme Gradient Boosting (XGBoost) achieved the highest accuracy of 96.4% and the highest precision score of 97%. The algorithm also demonstrated high scores for F-1, and recall, in comparison to the other two algorithms while achieving the lowest mislabeling score of 0.03%. This excellent performance can be attributed to the combination of decision trees, boosting, and regularization, which enables XGBoost to effectively capture complex relationships between features and avoid overfitting, leading to accurate prediction and low mislabeling. Linear Learner achieved an accuracy of 94.4% with similarly high precision, F-1, and recall, This suggests the algorithm's high performance and ability to handle large datasets effectively and make accurate predictions with a minimal mislabeling score of 0.05%. In contrast, k-Nearest Neighbor (k-NN) recorded the lowest accuracy among the three algorithms at 83.7% with a lower score for F-1 recall compared to the other two algorithms. However, the precision score of k-NN was high, indicating that the algorithm correctly predicted the positive samples.

Nevertheless, k-NN had the highest mislabeling score of 0.16% due to its difficulties in handling high-dimensional data.

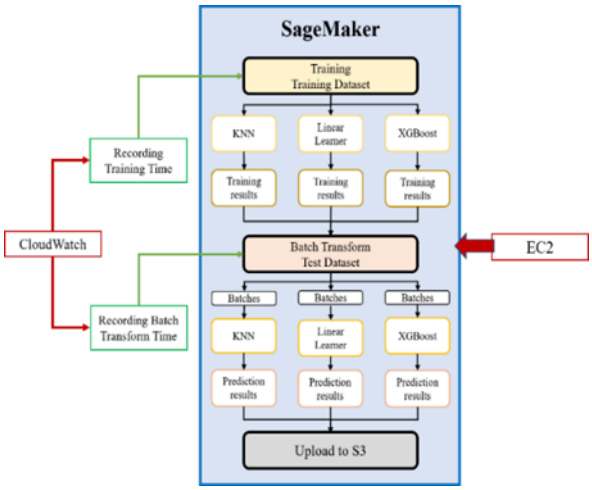

Fig. 9. Steps of training and evaluating the proposed method using Batch Transform

Table 2. evaluation metrics of the three algorithms after batch transform and endpoint prediction

Metrics	*Accuracy*	*Precision*	*F-1*	*Recall*	*Mislabeling*
XGBoost	0.96413	0.97086	0.96440	0.95802	0.03587
Linear Learner	0.94430	0.95263	0.94462	0.93675	0.05570
k-NN	0.83727	0.91150	0.82420	0.75216	0.16272

3.1.2. Endpoint Prediction Results

Endpoint is a part of the SageMaker Inference deployment procedure. Endpoint provides real-time predictions from a deployed model by allowing the developer to integrate the trained model into a workflow to make predictions on new data in real-time. The endpoint can be set up relatively quickly. It can host the resources of Cloud Machine Learning methods in Amazon SageMaker online for an infinite time until it is manually taken down. Endpoint can also offer scalable solutions to deploy more than one model and generate predictions from each model efficiently in real-time. This is done by enabling the time-sharing of memory and resources among the models and storing the artifacts in both Amazon Simple Storage Service (S3) bucket and the endpoint configuration, as shown in Fig. 10 which illustrates the steps of the batch transform process. In this study, the three algorithms were deployed using a multi-model Endpoint consisting of three created models, one for each algorithm. After Endpoint deployment, a prediction function was implemented to invoke the Endpoint by sending the test dataset to it and producing the prediction results. The evaluation results obtained from this prediction were the same as those obtained from the Batch Transform. However, the prediction time significantly decreased after the deployment as the prediction function was able to process the entire test dataset without the need to divide it into multiple records like Batch Transform and the Endpoint made it possible to have the results in real-time.

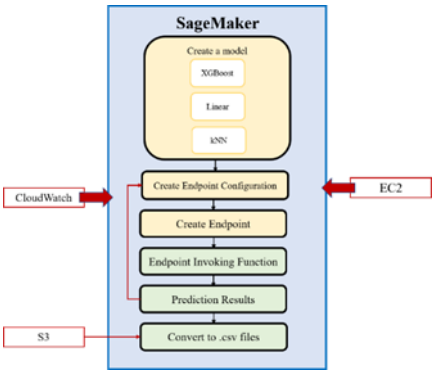

Fig. 10. Steps of creating and invoking the endpoint

3.2. Time Assessment Results

The training and Batch Transform times for the three algorithms used in the proposed method were calculated from Amazon CloudWatch logs. Based on the Amazon Elastic Compute Cloud (EC2) instance used for this method, Extreme Gradient Boosting (XGBoost) had the shortest training and Batch Transform times of 4.47 minutes and 5.38 minutes, respectively, due to its capability of handling big data and Multithreading Parallel Computing. Linear Learner had a decent training of 5.01 minutes and a Batch Transform time of 6.08 minutes, this is because it is optimized for use in distributed computing environments, allowing it to process large datasets quickly. K-Nearest Neighbor (k-NN) had the longest training time of 5.16 minutes and a Batch Transform time of 7.44 minutes, this is because k-NN works better with low dimensionality rather than a large number of features like in the dataset used in this study. Fig. 11 displays the prediction times of each algorithm, calculated from CloudWatch logs, and compares them to the training and Batch Transform times discussed previously. XGBoost had the shortest prediction time of only 0.0005 minutes, followed by Linear Learner, which had a prediction time of 0.0006 minutes, while k-NN had the longest prediction time of 0.0008 minutes. The reason behind the high speed of Endpoint prediction is that the deployed model was already loaded into memory, and the resources were available to perform predictions in real-time. When the endpoint received the prediction request, it used the loaded model to make the prediction, and the results were returned within milliseconds.

4. Conclusion

The study proposed a Cloud Machine Learning approach for identifying phishing websites using Amazon Web Service (AWS). The study utilized three AWS SageMaker-built-in

algorithms, namely, Extreme Gradient Boosting (XGBoost), Linear Learner, and k-Nearest Neighbor (k-NN), to detect phishing attacks using a dataset of 11,430 samples. The dataset was preprocessed, split into (70%) train and (30%) test datasets, and saved to an Amazon Simple Storage Service (S3) bucket. The preprocessing, training, and deployment resources were obtained from the Amazon Elastic Compute Cloud (EC2) instance (ml.m4.xlarge). Following training, the method was evaluated using SageMaker Batch Transform. The training and prediction times were calculated using Amazon CloudWatch. The XGBoost algorithm achieved the highest accuracy of 96.4%, with a training time of 4.47 minutes and a Batch Transform time of 5.38 minutes, followed by Linear Learner with an accuracy of 94.4%, a training time of 5.01 minutes and a Batch Transform time of 6.08 minutes. While k-NN achieved an accuracy of 83.7%, with a training time of 5.16 seconds and a Batch Transform time of 7.44 minutes. The method was deployed using a single SageMaker Inference multi-model Endpoint. After invoking the Endpoint with the prediction function, Amazon CloudWatch logs indicated that XGBoost outperformed the other algorithms, with the fastest prediction time at only 0.0005 minutes, followed by Linear Learner with 0.0006 minutes, and k-Nearest Neighbor with 0.0008 minutes. This approach can enable real-time prediction capabilities for applications that require near-instant response times, such as phishing detection systems. AWS provides efficient resources to handle large amounts of data in real-time, it is important to note the cost limitation of using such resources. Therefore, it is recommended to manually shut down all the resources after completing the project to avoid incurring additional costs. In the future, the accuracy and speed of the method can be improved by using a larger dataset and a different Amazon Elastic Compute Cloud (EC2) instance with more virtual resources.

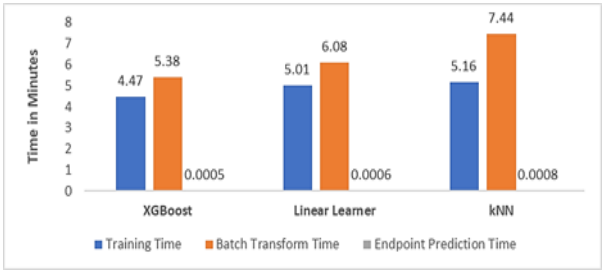

Fig. 11. Training, batch transform, and prediction times of the method

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] M. G. Cains, L. Flora, D. Taber, Z. King, and D. S. Henshel, "Defining Cyber Security and Cyber Security Risk within a Multidisciplinary Context using Expert Elicitation," Risk Analysis, 2021, doi: 10.1111/risa.13687.

[2] A. Mishra, Y. I. Alzoubi, M. J. Anwar, and A. Q. Gill, "Attributes impacting cybersecurity policy development: An evidence from seven nations," Comput Secur, vol. 120, p. 102820, Sep. 2022, doi: 10.1016/j.cose.2022.102820.

[3] A. A.A. and P. K., "Towards the Detection of Phishing Attacks," in 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), IEEE, Jun. 2020, pp. 337–343. doi: 10.1109/ICOEI48184.2020.9142967.

[4] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," Expert Syst Appl, vol. 117, pp. 345–357, Mar. 2019, doi: 10.1016/j.eswa.2018.09.029.

[5] Md. S. I. Islam Prottasha, Md. Z. Rahman, A. K. Hossain, S. F. Mou, Md. B. Ahmed, and M. S. Kaiser, "Vote algorithm based probabilistic model for phishing website detection," Indonesian Journal of Electrical Engineering and Computer Science, vol. 28, no. 3, p. 1582, Dec. 2022, doi: 10.11591/ijeecs.v28.i3.pp1582-1591.

[6] M. Korkmaz, O. K. Sahingoz, and B. Diri, "Detection of Phishing Websites by Using Machine Learning-Based URL Analysis," in 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, Jul. 2020, pp. 1–7. doi: 10.1109/ICCCNT49239.2020.9225561.

[7] J. Kumar, A. Santhanavijayan, B. Janet, B. Rajendran, and B. S. Bindhumadhava, "Phishing Website Classification and Detection Using Machine Learning," in 2020 International Conference on Computer Communication and Informatics (ICCCI), IEEE, Jan. 2020, pp. 1–6. doi: 10.1109/ICCCI48352.2020.9104161.

[8] C. Gu, "A Lightweight Phishing Website Detection Algorithm by Machine Learning," in 2021 International Conference on Signal Processing and Machine Learning (CONF-SPML), IEEE, Nov. 2021, pp. 245–249. doi: 10.1109/CONF-SPML54095.2021.00054.

[9] N. S. Zaini et al., "Phishing detection system using nachine learning classifiers," Indonesian Journal of Electrical Engineering and Computer Science, vol. 17,

no. 3, p. 1165, Mar. 2020, doi: 10.11591/ijeecs.v17.i3.pp1165-1171.

[10] Y. Wei and Y. Sekiya, "Sufficiency of Ensemble Machine Learning Methods for Phishing Websites Detection," IEEE Access, vol. 10, pp. 124103–124113, 2022, doi: 10.1109/ACCESS.2022.3224781.

[11] S. Das Guptta, K. T. Shahriar, H. Alqahtani, D. Alsalman, and I. H. Sarker, "Modeling Hybrid Feature-Based Phishing Websites Detection Using Machine Learning Techniques," Annals of Data Science, Mar. 2022, doi: 10.1007/s40745-022-00379-8.

[12] A. Nandi Tultul, R. Afroz, and M. A. Hossain, "Comparison of the efficiency of machine learning algorithms for phishing detection from uniform resource locator," Indonesian Journal of Electrical Engineering and Computer Science, vol. 28, no. 3, p. 1640, Dec. 2022, doi: 10.11591/ijeecs.v28.i3.pp1640-1648.

[13] A. A. Orunsolu, A. S. Sodiya, and A. T. Akinwale, "A predictive model for phishing detection," Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 2, pp. 232–247, Feb. 2022, doi: 10.1016/j.jksuci.2019.12.005.

[14] I. Saha, D. Sarma, R. J. Chakma, M. N. Alam, A. Sultana, and S. Hossain, "Phishing attacks detection using deep learning approach," in Proceedings of the 3rd International Conference on Smart Systems and Inventive Technology, ICSSIT 2020, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 1180–1185. doi: 10.1109/ICSSIT48917.2020.9214132.

[15] P. Mccaffrey, "Cloud technologies," in An Introduction to Healthcare Informatics, Elsevier, 2020, pp. 307–316. doi: 10.1016/B978-0-12-814915-7.00021-1.

[16] K. Swedha and T. Dubey, "Analysis of Web Authentication Methods Using Amazon Web Services," in 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, Jul. 2018, pp. 1–6. doi: 10.1109/ICCCNT.2018.8494054.

[17] H. Singh, Practical Machine Learning with AWS. Berkeley, CA: Apress, 2021. doi: 10.1007/978-1-4842-6222-1.

[18] E. S. Gualberto, R. T. de Sousa, T. P. B. de Vieira, J. P. C. L. da Costa, and C. G. Duque, "From Feature Engineering and Topics Models to Enhanced Prediction Rates in Phishing Detection," IEEE Access, vol. 8, pp. 76368–76385, 2020, doi: 10.1109/ACCESS.2020.2989126.

[19] X. Y. Liew, N. Hameed, and J. Clos, "An investigation of XGBoost-based algorithm for breast cancer classification," Machine Learning with Applications, vol. 6, p. 100154, Dec. 2021, doi: 10.1016/j.mlwa.2021.100154.

[20] M. Chen, Q. Liu, S. Chen, Y. Liu, C. H. Zhang, and R. Liu, "XGBoost-Based Algorithm Interpretation and Application on Post-Fault Transient Stability Status Prediction of Power System," IEEE Access, vol. 7, pp. 13149–13158, 2019, doi: 10.1109/ACCESS.2019.2893448.

[21] J. Cao, J. Gao, H. Nikafshan Rad, A. S. Mohammed, M. Hasanipanah, and J. Zhou, "A novel systematic and evolved approach based on XGBoost-firefly algorithm to predict Young's modulus and unconfined compressive strength of rock," Eng Comput, vol. 38, no. S5, pp. 3829–3845, Dec. 2022, doi: 10.1007/s00366-020-01241-2.

[22] X. Y. Liew, N. Hameed, and J. Clos, "An investigation of XGBoost-based algorithm for breast cancer classification," Machine Learning with Applications, vol. 6, p. 100154, Dec. 2021, doi: 10.1016/j.mlwa.2021.100154.

[23] Aishwarya V, "Binary Classification Model for Fraudulent Credit Card Transactions," IOSR J Comput Eng, vol. 22, no. 3, pp. 38–45, 2020, doi: 10.9790/0661-2203023845.

[24] N. F. Abedin, R. Bawm, T. Sarwar, M. Saifuddin, M. A. Rahman, and S. Hossain, "Phishing Attack Detection using Machine Learning Classification Techniques," in 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), IEEE, Dec. 2020, pp. 1125–1130. doi: 10.1109/ICISS49785.2020.9315895.

[25] M. Rastogi, A. Chhetri, D. K. Singh, and G. Rajan V, "Survey on Detection and Prevention of Phishing Websites using Machine Learning," in 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), IEEE, Mar. 2021, pp. 78–82. doi: 10.1109/ICACITE51222.2021.9404714.

[26] A. Niranjan, D. K. Haripriya, R. Pooja, S. Sarah, P. Deepa Shenoy, and K. R. Venugopal, "EKRV: Ensemble of kNN and Random Committee Using Voting for Efficient Classification of Phishing," in Advances in Intelligent Systems and Computing, Springer Verlag, 2019, pp. 403–414. doi: 10.1007/978-981-13-1708-8_37.

[27] R. D. Prayogo and S. A. Karimah, "Optimization of Phishing Website Classification Based on Synthetic Minority Oversampling Technique and Feature Selection," in 2020 International Workshop on Big Data and Information Security (IWBIS), IEEE, Oct. 2020, pp. 121–126. doi: 10.1109/IWBIS50925.2020.9255562.