Phishing Website Detection by Machine Learning Techniques

Karan Vora (kv2154) kv2154@nyu.edu
Department of Electrical and Computer Engineering
New York University

Aman Mittal (am11982) am11982@nyu.edu
Department of Electrical and Computer Engineering
New York University

Siddharth Shah (ss16130) ss16130@nyu.edu
Department of Electrical and Computer Engineering
New York University

Parth Metha (pjm9767) pjm9767@nyu.edu
Department of Electrical and Computer Engineering
New York University

Code and Files

The code and files for this project are available at Final Project.

Abstract

Phishing is a type of Internet fraud in which a malicious actor sends out fake messages that appear to come from a reliable source. These messages often contain a link or file that, when clicked, can steal personal information or infect a computer with a virus. In the past, phishing attempts were conducted through large-scale spam campaigns that indiscriminately targeted a wide range of people. The goal was to get as many people as possible to click on a link or open an infected file. To detect this type of attack, one approach is to use machine learning. The URLs received by the user are fed into a machine learning model, which then processes the input and displays an output indicating whether the URL is phishing or legitimate. Various ML algorithms such as SVM, Neural Networks, Random Forest, Decision Tree, and XG boost can be used to classify these URLs.

Introduction

In today's digital age, marked by an extensive dependence on technology, online activities are deeply intertwined with our professional and personal lives. Unfortunately, this shift has been paralleled by a significant rise in cybercrime, with phishing becoming a predominant threat. Phishing, a serious security concern, results in substantial financial and data loss for both businesses and individuals. The growing frequency of these attacks, exacerbated by inadequate detection and prevention methods, demands immediate and effective countermeasures. Ensuring the security of internet users against phishing requires the establishment of a comprehensive and potent detection strategy to preserve data integrity

and privacy. In this landscape, machine learning emerges as a viable solution, adept at recognizing and mitigating such cyber threats.

Phishing, a cybercrime characterized by the deceptive gathering of sensitive data via apparent legitimate communication channels such as emails, phone calls, or text messages, presents a unique set of challenges. Conventional approaches, such as updating antivirus databases with blacklisted URLs and IPs, fall short, especially in countering zero-hour phishing assaults. Heuristic-based detection systems, designed to identify zero-hour attacks based on typical phishing traits, are plagued by high false-positive rates and variable characteristics in different phishing scenarios. To address these shortcomings, the adoption of machine learning technologies is advocated. This suite of algorithms, trained on historical data, is capable of making informed predictions about new threats. By examining a range of legitimate and phishing URLs, machine learning algorithms proficiently distinguish and highlight phishing sites, encompassing those involved in zero-hour attacks. This method signifies a significant advancement in combating cybercrime, enhancing both accuracy and adaptability in phishing detection.

Addressing the substantial risk of user data exploitation posed by phishing remains a critical and complex challenge. Recent research has centered on analyzing domain attributes such as website URLs and content, including source code and website screenshots, to thwart phishing. Despite these efforts, there is a noticeable gap in efficient anti-phishing tools for organizational defense. The presence of malicious code on websites poses serious risks for data theft and malware proliferation, intensifying cybersecurity and privacy concerns. Machine Learning (ML) techniques have proven effective in detecting malicious URLs. Traditional detection methods, reliant on user-reported or expertly analyzed blacklists of known malicious URLs, are increasingly inadequate due to the continual emergence of unlisted malicious URLs and cybercriminals' use of Domain Generation Algorithms (DGA) to bypass these lists. Thus, ML-based strategies have been proposed to surmount the limitations inherent in blacklist-based systems. Regarded as a binary classification task, ML training involves mapping a multi-dimensional vector space to a binary outcome, differentiating between malicious and benign URLs. This strategy boasts a robust generalization ability, enabling it to identify new malicious URLs, an achievement beyond the reach of conventional blacklist methods. Mandadi et al. (2022)

Literature Survey

In this review of the literature, we examine the corpus of work that has already been done on the categorization of phishing URLs and the use of deep learning and machine learning methods in the field of cybersecurity. Gaining knowledge of the approaches, difficulties, and developments in the field will help to lay the groundwork for the current research, which focuses on robust URL classification. Alnemari and Alshammari (2023) compares the use of artificial neural networks (ANNs), support vector machines (SVMs), decision trees (DTs), and random forest (RF) techniques to detect phishing domains. Sönmez et al. (2018) displays a study to perform Extreme Learning Machine (ELM) based classification for 30 features including Phishing Websites Data in UC Irvine Machine Learning Repository database. For results assessment, ELM was compared with other machine learning methods such as Support Vector Machine (SVM), Naïve Bayes (NB). Mandadi et al. (2022) suggests

the use of system that learns about approaches, especially supervised mastering. The XG-Boost method's genuine overall performance in classification is the reason they chose it. By evaluating the characteristics of phishing websites and selecting the best combination of them to train the classifier, their goal was to develop a higher overall performance classifier. C.Iwendi and O.Jo (2020) shows the development of an algorithm that adds watermark logically into the code utilizing the inherent properties of code and gives a robust solution. The embedding algorithm uses keywords to make segments of the code to produce a key-dependent on the watermark. The extraction algorithms use this key to remove watermark and detect tampering. Snelling (2014) describes technology by which phishing-related data sources are processed into aggregated data and a given site evaluated the aggregated data using a predictive model to automatically determine whether the given site is likely to be a phishing site. Erzhou Zhu and Fang (2020) paper proposes to use DTOF-ANN (Decision Tree and Optimal Features based Artificial Neural Network) to tackle this shortcoming, which is a neural-network phishing detection model based on decision tree and optimal feature selection. Abdulhamit Subasi and Chaudhery (2017) proposes an intelligent system to detect phishing attacks using different data mining techniques to decide categories of websites: legitimate or phishing. Different classifiers were used in order to construct accurate intelligent system for phishing website detection. Classification accuracy, area under receiver operating characteristic (ROC) curves (AUC) and F-measure was used to evaluate the performance of the data mining techniques. Results showed that Random Forest was the best classification method to deal with different websites in phishing detection.

Methodology and Proposed Approach

Our methodology employs a comprehensive and multifaceted approach, utilizing an array of algorithms and datasets meticulously curated to optimize and derive the most effective solution for phishing detection. Central to this approach is an end-to-end solution architecture, encompassing a series of interconnected and sophisticated pipelines.

At the outset, we have established a robust Data-Processing pipeline. This integral component of our methodology is designed to meticulously handle and prepare vast datasets, ensuring they are optimally formatted for subsequent analysis. The pipeline involves intricate processes such as data cleansing, normalization, feature extraction, and transformation. These steps are crucial in refining the raw data into a structured and analyzable form, laying a solid foundation for the accuracy and efficiency of the machine learning models.

Following the data processing stage, we transition into the core of our methodology - the Machine Learning and Neural Network training pipeline. This stage is characterized by the application of advanced machine learning algorithms, encompassing both traditional methods and cutting-edge neural network techniques. The algorithms employed are selected and fine-tuned to cater specifically to the nuances and complexities inherent in phishing detection. This stage involves rigorous training, validation, and optimization cycles, utilizing cross-validation techniques and hyperparameter tuning to enhance the model's performance and ensure its reliability in accurately identifying phishing attempts.

Finally, the culmination of our methodology is the deployment of the final model. This phase marks the transition of our solution from a theoretical framework to a practical, user-oriented tool. The model is integrated into a browser extension, a strategic choice that

positions our solution directly in the user's line of defense against phishing attacks. This browser extension serves as a real-time phishing detection tool, seamlessly analyzing visited URLs and alerting users of potential threats. The extension is designed to be lightweight, non-intrusive, and user-friendly, ensuring it provides effective security measures without compromising the user experience.

In summary, our methodology represents a holistic and dynamic approach to phishing detection. By combining a meticulous data-processing pipeline with sophisticated machine learning and neural network training, and culminating in the practical deployment of a browser extension, we offer a solution that is not only theoretically sound but also pragmatically effective in safeguarding users against the ever-present threat of phishing in the digital world.

- **Decision Tree**
  Decision tree learning employs a divide-and-conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class. Quinlan (2014), Hastie et al. (2009)

  - **Node Splitting**
    * Each node in a decision tree represents a feature (attribute) of the dataset.
    * The decision of how to split a node is based on metrics such as Information Gain, Gini Impurity, or Mean Squared Error (for regression trees).
  - **Information Gain**
    * Information Gain is used to measure how much a feature contributes to reducing uncertainty or entropy in the target variable. Essentially, it quantifies the effectiveness of an attribute in classifying a dataset. Information Gain measures the reduction in entropy before and after the dataset is split on an attribute. Essentially, it tells us how much uncertainty in the target variable is reduced after observing an attribute.
    * Entropy of a dataset is a measure of the amount of uncertainty in the dataset (S) and is given by the equation:

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

      where p_i is the proportion of the number of elements in class i to the number of elements in set S, and c is the number of classes.
    * Information Gain is calculated as:

$$\text{IG}(S, A) = \text{Entropy}(S) - \sum_{t \in T} \frac{|S_t|}{|S|} \text{Entropy}(S_t)$$

where A is the attribute (feature), T are the subsets created from splitting set S by attribute A, and $|S_t|$ and $|S|$ are the sizes of subset t and set S, respectively.

- **Gini Impurity**
  - Gini Impurity is a metric used in machine learning, particularly in decision tree algorithms like CART (Classification and Regression Trees), to measure the degree of impurity or disorder in a set of data. It plays a crucial role in the decision-making process of the tree, helping to determine how a dataset should be split at each node. Gini Impurity quantifies the likelihood of incorrect classification of an element if it were randomly labeled according to the distribution of labels in the subset. A Gini Impurity of 0 indicates that all elements in the subset belong to a single class, implying perfect purity.
  - Gini Impurity is calculated as:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2$$

where p_i is the proportion of the number of elements in class i to the number of elements in set S.

- **Stopping Criteria and Prunning**
  - Stopping criteria are the rules or conditions defined to stop the further splitting of decision tree nodes. These criteria are essential to prevent the tree from becoming too complex and overfitting the data. Overfitting occurs when the tree is too detailed, capturing noise in the data rather than the actual signal.
  - Pruning is a technique used to reduce the size of a decision tree after it has been built. It is a crucial step to enhance the model's ability to generalize to unseen data. Pruning removes parts of the tree that may be capturing noise, thereby simplifying the model.
- **Prediction**
  - Prediction in a Decision Tree refers to the process of using the tree to forecast the outcome for a new instance based on the learned patterns from the training data. This process involves traversing the tree from the root to a leaf, following the path determined by the values of the instance's attributes. The outcome at the leaf node provides the prediction.

- **Support Vector Machine (SVM)**
  Support Vector Machine (SVM) is a type of supervised machine learning algorithm primarily used for classification tasks, though it can also be applied to regression. The core idea behind SVM is to find the best-separating hyperplane that divides data points from different classes in the feature space. Cortes and Vapnik (1995), Vapnik (1998), Schölkopf and Smola (2002)
  - **Hyperplane**

* In SVM, a hyperplane is a decision boundary separating different classes. For a 2-dimensional dataset, this hyperplane is a line, while in 3 dimensions, it's a plane, and so forth for higher dimensions.
* A hyperplane in an n-dimensional space is defined as the set of points $\mathbf{x}$ satisfying:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane and b is the bias term.

– **Decision Rule**

* For a binary classification, the decision function is:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

. A data point $\mathbf{x}$ is classified into one of two classes based on the sign of this function.

– **Margin and Support Vectors**

* The margin is defined as the distance between the hyperplane and the nearest data point from either class. SVM seeks to maximize this margin, which contributes to the model's generalization ability.
* Support vectors are the data points that are closest to the hyperplane. These points are critical in defining the position and orientation of the hyperplane. The SVM algorithm focuses on these points, ignoring others that are further away from the decision boundary.

– **Optimization Problem**

* The SVM training algorithm builds a model that assigns new unseen data points into one category or the other.
* The optimization problem can be formulated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$$

where y_i are the labels of the training examples.

– **Kernel**

* VM can employ various kernel functions (like polynomial, radial basis function, or sigmoid kernels) to transform the input space into a higher-dimensional space where it's easier to separate the data linearly. This is particularly useful for handling complex datasets where linear separation in the original space is not possible.

- **Random Forest**
The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision

trees. Feature randomness, also known as feature bagging or "the random subspace method", generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features. Breiman (2001), Hastie et al. (2009), Ho (1998)

- **Ensemble learning and Bagging**
  - Ensemble learning methods are made up of a set of classifiers—e.g. decision trees—and their predictions are aggregated to identify the most popular result. The most well-known ensemble methods are bagging, also known as bootstrap aggregation, and boosting. In 1996, Leo Breiman introduced the bagging method; in this method, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these models are then trained independently, and depending on the type of task—i.e. regression or classification—the average or majority of those predictions yield a more accurate estimate. This approach is commonly used to reduce variance within a noisy dataset.
  - Each tree in a Random Forest is built from a sample drawn with a replacement (bootstrap sample) from the training set. For a training set D of size N, a sample is drawn with replacement to create a subset of the same size. This subset is used to train a decision tree.
- **Feature Randomness**
  - When building each tree, at each split, a random subset of features is chosen. This ensures that the trees in the forest are diverse. If there are M features, a number $m \ll M$ is specified such that at each node, m features are randomly selected, and the best split on these m features is used to split the node. This parameter is key and needs to be chosen carefully.
- **Prediction**
  - Classification: For a classification task, each tree in the forest outputs a class prediction, and the class with the most votes becomes the model's prediction.
  - Regression: For regression, the average of all the tree outputs is considered as the final prediction.
- **Equation of Prediction**
  - Let $h(x, \Theta_k)$ be the prediction of the k-th tree for input x. Θ_k represents the randomness in the tree construction. The Random Forest prediction $H(x)$ is given by: Classification (Majority Voting):

$$H(x) = \text{mode}\{h(x, \Theta_1), h(x, \Theta_2), ..., h(x, \Theta_K)\}$$

Regression (Average):

$$H(x) = \frac{1}{K} \sum_{k=1}^{K} h(x, \Theta_k)$$

where K is the number of trees.

- **Naive Bayes**
  Naïve Bayes classifiers work differently in that they operate under a couple of key assumptions, earning it the title of "naïve". It assumes that predictors in a Naïve Bayes model are conditionally independent, or unrelated to any of the other features in the model. It also assumes that all features contribute equally to the outcome. While these assumptions are often violated in real-world scenarios (e.g. a subsequent word in an e-mail is dependent upon the word that precedes it), it simplifies a classification problem by making it more computationally tractable. That is, only a single probability will now be required for each variable, which, in turn, makes the model computation easier. Despite this unrealistic independence assumption, the classification algorithm performs well, particularly with small sample sizes. McCallum and Nigam (1998), Rish (2001), Manning et al. (2008)

  - **Bayes' Theorem** Naive Bayes relies on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. The theorem is stated as:

    $$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

    where A and B are events, $P(A|B)$ is the probability of A given B, $P(B|A)$ is the probability of B given A, $P(A)$ is the probability of A, and $P(B)$ is the probability of B.
  - **Applying to Classification** In a classification task, A represents a class label, and B represents the features. The goal is to find the class C that maximizes $P(C|X)$, where X is the feature vector.
  - **Assumption of Feature Independence** Naive Bayes simplifies computation by assuming that features are conditionally independent given the class label. This means:

    $$P(X|C) = P(x_1, x_2, ..., x_n|C) = \prod_{i=1}^{n} P(x_i|C)$$

    where $X = (x_1, x_2, ..., x_n)$ is the feature vector.
  - **Classification Rule** The classifier predicts the class C_k for a given feature vector X that maximizes $P(C_k|X)$:

    $$\hat{y} = \arg\max_{C_k} P(C_k) \prod_{i=1}^{n} P(x_i|C_k)$$

    Here, $P(C_k)$ is the prior probability of class C_k, and $P(x_i|C_k)$ is the likelihood of feature i given class C_k.
  - **Model Training** In training, the algorithm estimates the probabilities $P(C_k)$ and $P(x_i|C_k)$ from the training data.

- **K-Nearest Neighbors (KNN)**
  The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric,

supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. Cover and Hart (1967), Duda et al. (2012), Altman (1992)

- **Distance Metric** In both classification and regression, k-NN works by finding the k nearest data points to the input point and making predictions based on these nearest neighbors. The algorithm calculates the distance between points using a distance metric, typically Euclidean distance in a two-dimensional space, given by:

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

  where $\mathbf{p}$ and $\mathbf{q}$ are two points in Euclidean n-space, and p_i and q_i are coordinates of $\mathbf{p}$ and $\mathbf{q}$ respectively. Other distance metrics like Manhattan, Minkowski, or Hamming distance can also be used depending on the type of data.

- **Classification Rule** In a classification task, the algorithm assigns the input point the most common class among its k nearest neighbors. The prediction is often made by majority voting:

$$\hat{y} = \text{mode}\{y_{i1}, y_{i2}, \ldots, y_{ik}\}$$

  where y_{ij} is the class label of the j-th nearest neighbor of the input point.

- **Regression Rule** For regression, k-NN predicts the output for the input point based on the average of the values of its k nearest neighbors. The prediction is given by:

$$\hat{y} = \frac{1}{k} \sum_{j=1}^{k} y_{ij}$$

  where y_{ij} is the value of the j-th nearest neighbor.

- **Choosing k** The choice of k is critical and is usually selected via cross-validation. A small value of k leads to a model that is highly sensitive to noise, while a large k makes the algorithm computationally expensive and might include points from other classes.

- **Neural Networks**

Classification using a dense (or fully connected) neural network involves training a network of interconnected neurons to classify input data into categories. This process is grounded in the foundational concepts of neural network architecture, activation functions, and backpropagation for training. Goodfellow et al. (2016), LeCun et al. (2015), Rumelhart et al. (1986)

- **Network Architecture** A dense neural network consists of layers of neurons: an input layer, several hidden layers, and an output layer. Each neuron in a layer is connected to all neurons in the previous and next layers.

– **Neuron Computation** The output of each neuron is computed as a weighted sum of its inputs, plus a bias term, followed by an activation function. Mathematically, for a neuron j in layer l, this is:

$$a_j^{(l)} = \sigma\left(\sum_i w_{ji}^{(l)} \cdot a_i^{(l-1)} + b_j^{(l)}\right)$$

where $a_i^{(l-1)}$ are activations from neurons in the previous layer, $w_{ji}^{(l)}$ are weights, $b_j^{(l)}$ is the bias, and σ is the activation function.

– **Activation Functions** Common activation functions include the sigmoid (for binary classification), softmax (for multi-class classification in the output layer), ReLU (for hidden layers), and tanh. For example, the softmax function for a neuron j in the output layer is:

$$\text{softmax}(a_j) = \frac{e^{a_j}}{\sum_k e^{a_k}}$$

where a_j is the neuron's activation, and the denominator is the sum of the exponential activations of all neurons in the output layer.

– **Loss Function** The loss function measures the difference between the network's predictions and the actual labels. Common loss functions include cross-entropy for classification:

$$\text{Cross-Entropy}(y, \hat{y}) = -\sum y \log(\hat{y})$$

where y is the true label, and $\hat{y}$ is the predicted probability.

– **Backpropagation and Training** Backpropagation is used to calculate the gradient of the loss function with respect to each weight in the network. Weights are then updated using an optimization algorithm like Gradient Descent, where:

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \eta \frac{\partial \text{Loss}}{\partial w_{ji}^{(l)}}$$

where η is the learning rate.

Results and plots

In evaluating the efficacy of machine learning algorithms for classification tasks, researchers commonly employ a suite of performance metrics and visualization techniques to comprehensively analyze model behavior. The Support Vector Machine (SVM), a widely recognized classification algorithm, is often subject to such rigorous scrutiny Cortes and Vapnik (1995). The performance of SVM, as with other algorithms, is typically assessed using a combination of accuracy metrics across training, validation, and testing phases, as well as through more granular measures such as precision, recall, and the F1 score Bishop (2006).

Accuracy metrics serve as a primary indicator of model performance, with training accuracy reflecting the model's fit to the data upon which it was trained. Validation accuracy, typically obtained through k-fold cross-validation procedures, provides insight into

the model's ability to generalize beyond the training dataset to unseen data Kohavi (1995). Testing accuracy, derived from the model's performance on a distinct and previously untouched dataset, offers an unbiased assessment of the model's generalizability. Discrepancies between these metrics can be indicative of overfitting if the training accuracy significantly surpasses the validation and test accuracies James et al. (2013).

Cross-validation, a robust statistical method to estimate the skill of machine learning models, involves partitioning the dataset into complementary subsets, performing the analysis on one subset (the training set), and validating the analysis on the other subset (the validation set) Arlot and Celisse (2010). A visualization of model accuracy across k-folds allows for an assessment of the model's stability and variance, providing confidence in the results' reliability.

Beyond aggregate accuracy, performance metrics such as precision, recall, and the F1 score offer nuanced views of model efficacy Powers (2011). Precision, defined as the ratio of true positives to the sum of true and false positives, measures the exactness of the classifier, whereas recall, the ratio of true positives to the sum of true positives and false negatives, measures the classifier's completeness. The F1 score, the harmonic mean of precision and recall, serves as a single metric that balances the trade-off between precision and recall, especially in situations of class imbalance Van Rijsbergen (1979).

The confusion matrix, a tabular visualization of a classifier's predictions against the true labels, further dissects model performance, providing insights into the types of errors made by the classifier Stehman (1997). An analysis of the confusion matrix can uncover tendencies in misclassification, such as a propensity to incorrectly predict a specific class, which may suggest biases in the model or the data.

Plot for Decision Tree on Dataset 1

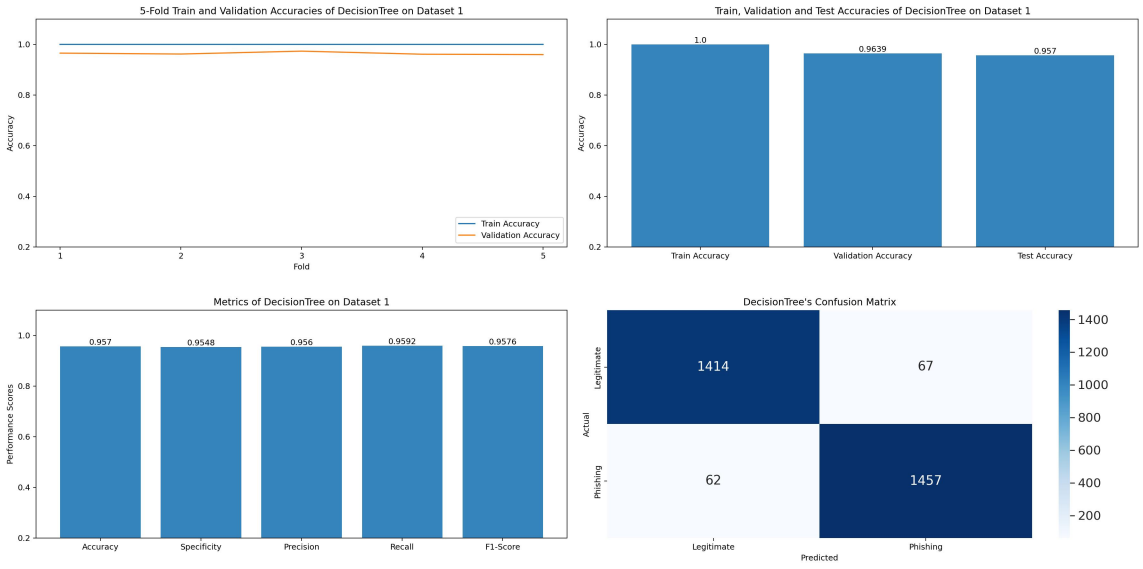

Figure 1: Plot for Decision Tree on Dataset 1

Plot for Decision Tree on Dataset 2

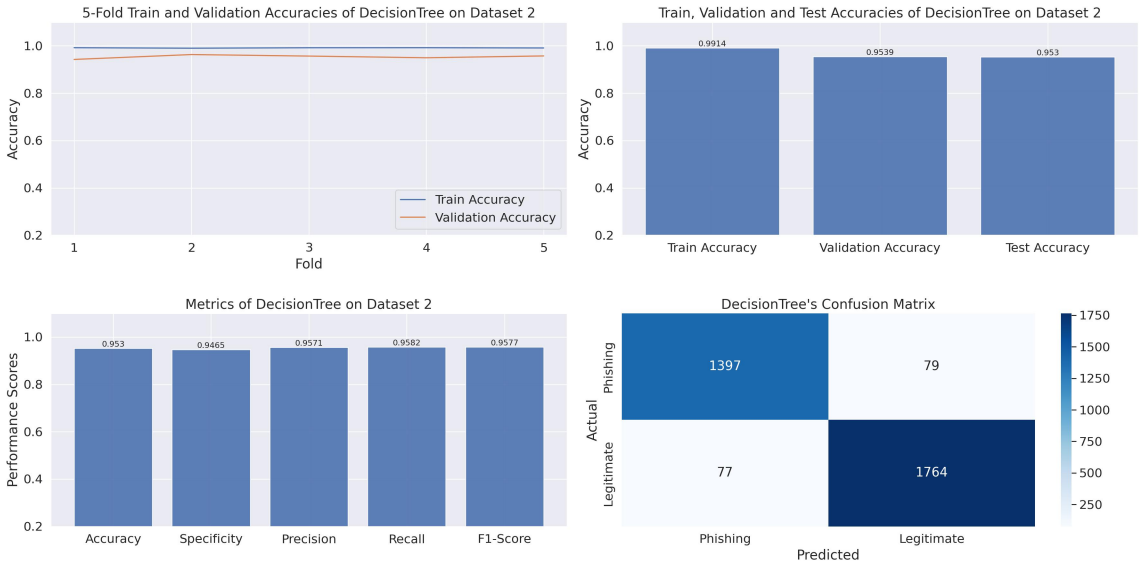

Figure 2: Plot for Decision Tree on Dataset 2

Plot for Decision Tree on Dataset 3

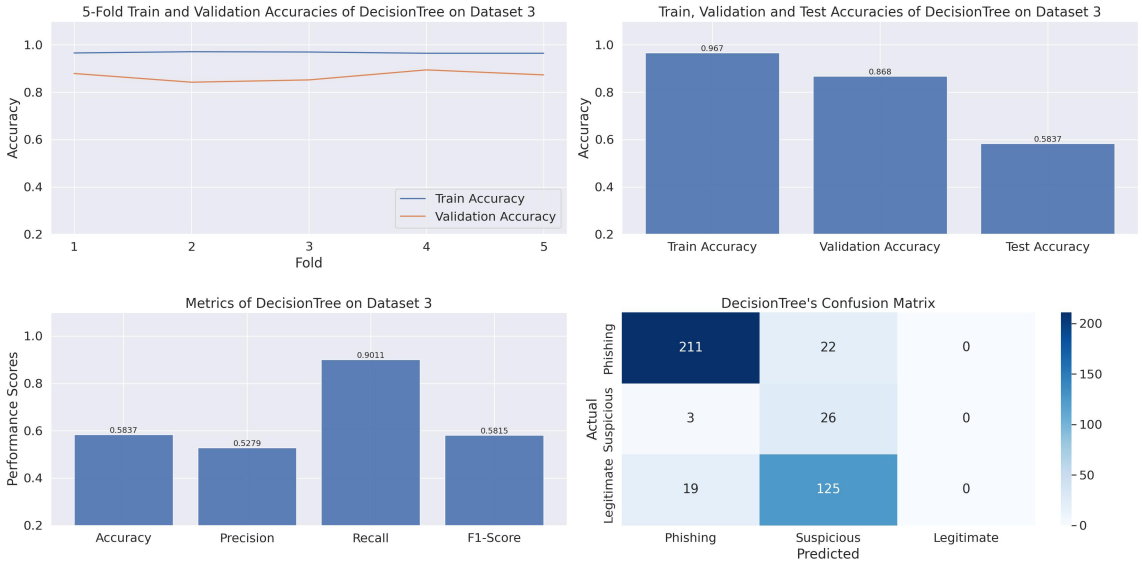

Figure 3: Plot for Decision Tree on Dataset 3

Plot for Support Vector Machine (SVM) on Dataset 1

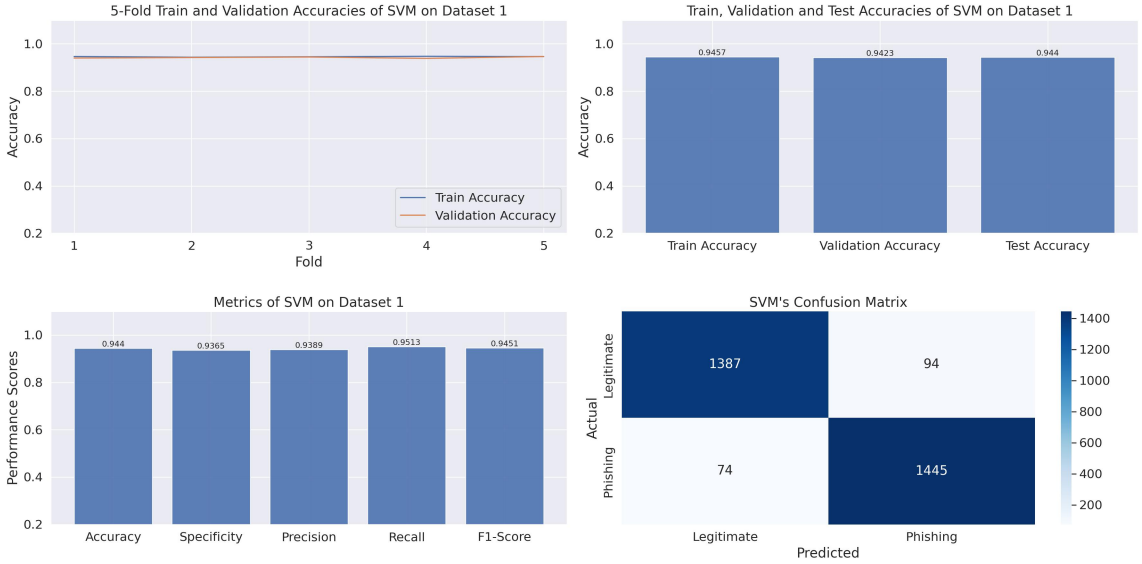

Figure 4: Plot for Support Vector Machine (SVM) on Dataset 1

Plot for Support Vector Machine (SVM) on Dataset 2

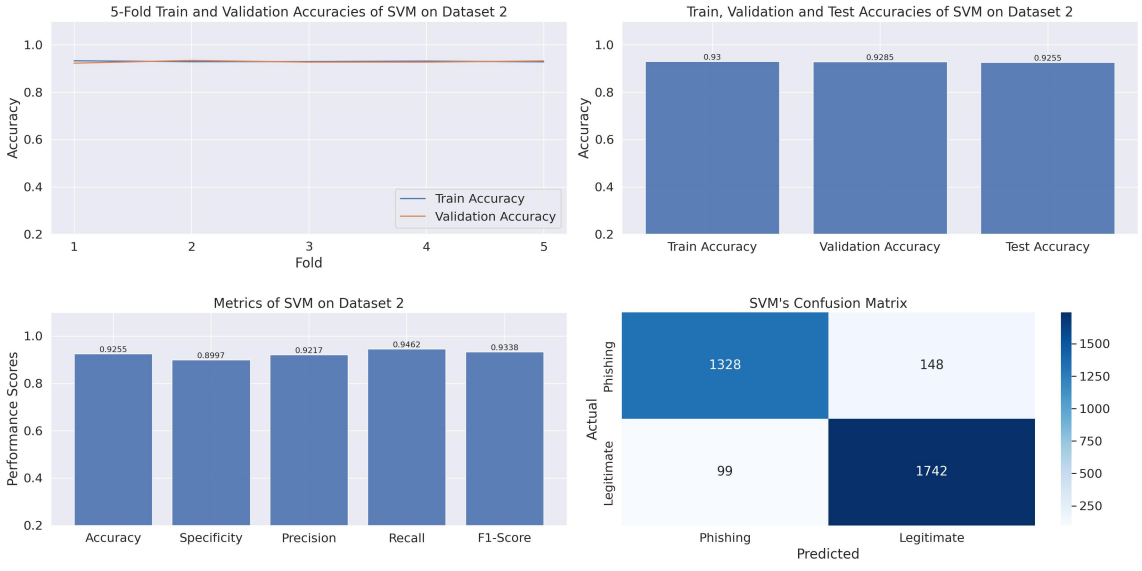

Figure 5: Plot for Support Vector Machine (SVM) on Dataset 2

Plot for Support Vector Machine (SVM) on Dataset 3

Figure 6: Plot for Support Vector Machine (SVM) on Dataset 3

Plot for Random Forest on Dataset 1

Figure 7: Plot for Random Forest on Dataset 1

Plot for Random Forest on Dataset 2

Figure 8: Plot for Random Forest on Dataset 2

Plot for Random Forest on Dataset 3

Figure 9: Plot for Random Forest on Dataset 3

Plot for Naive Bayes on Dataset 1

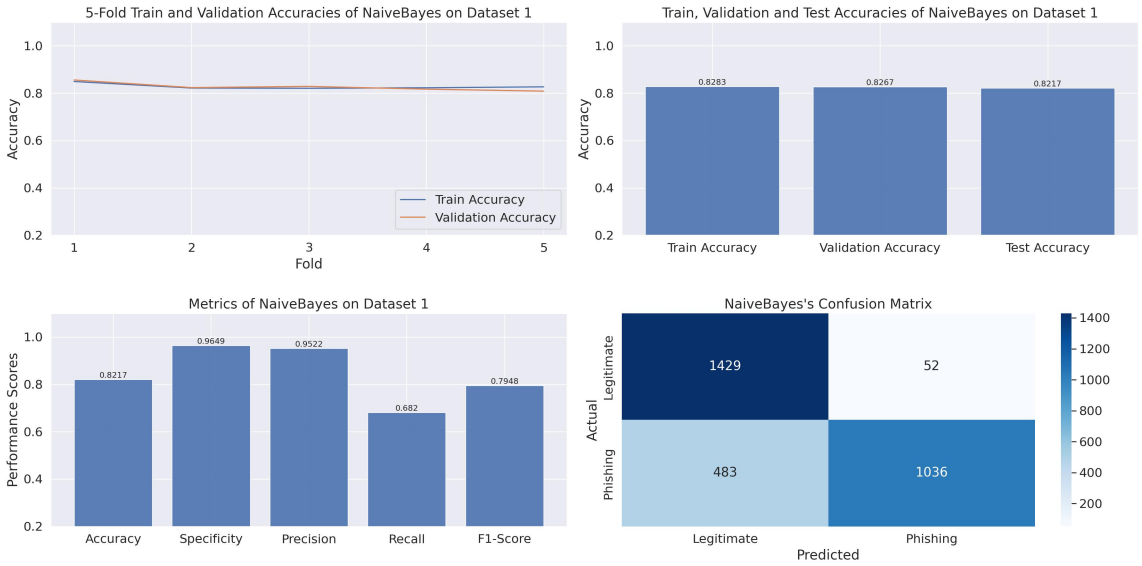

Figure 10: Plot for Naive Bayes on Dataset 1

Plot for Naive Bayes on Dataset 2

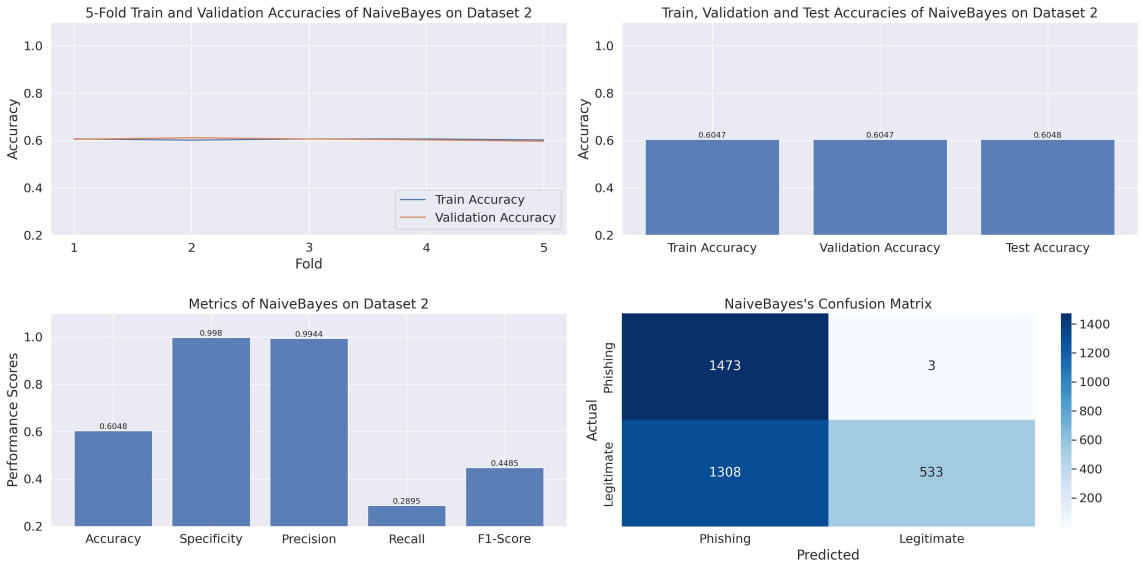

Figure 11: Plot for Naive Bayes on Dataset 2

Plot for Naive Bayes on Dataset 3

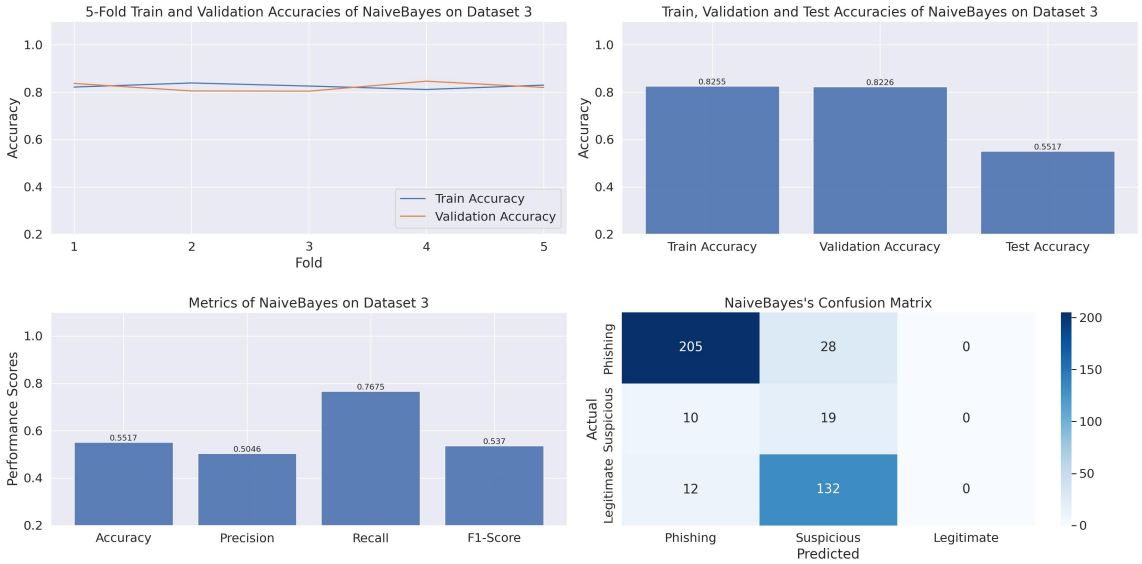

Figure 12: Plot for Naive Bayes on Dataset 3

Plot for K-Nearest Neighbors (KNN) on Dataset 1

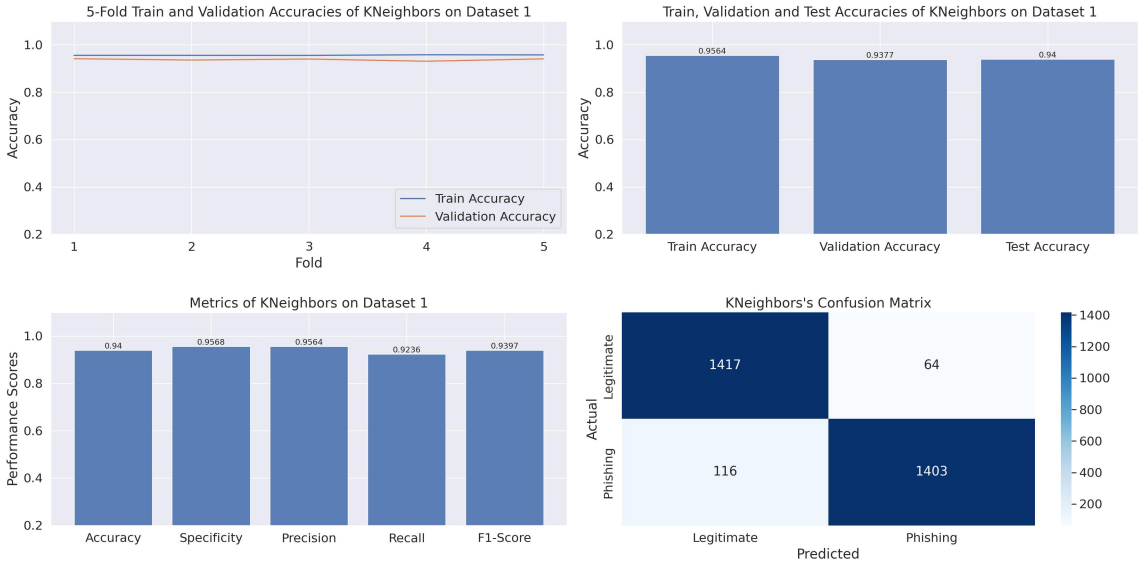

Figure 13: Plot for K-Nearest Neighbors (KNN) on Dataset 1

Plot for K-Nearest Neighbors (KNN) on Dataset 2

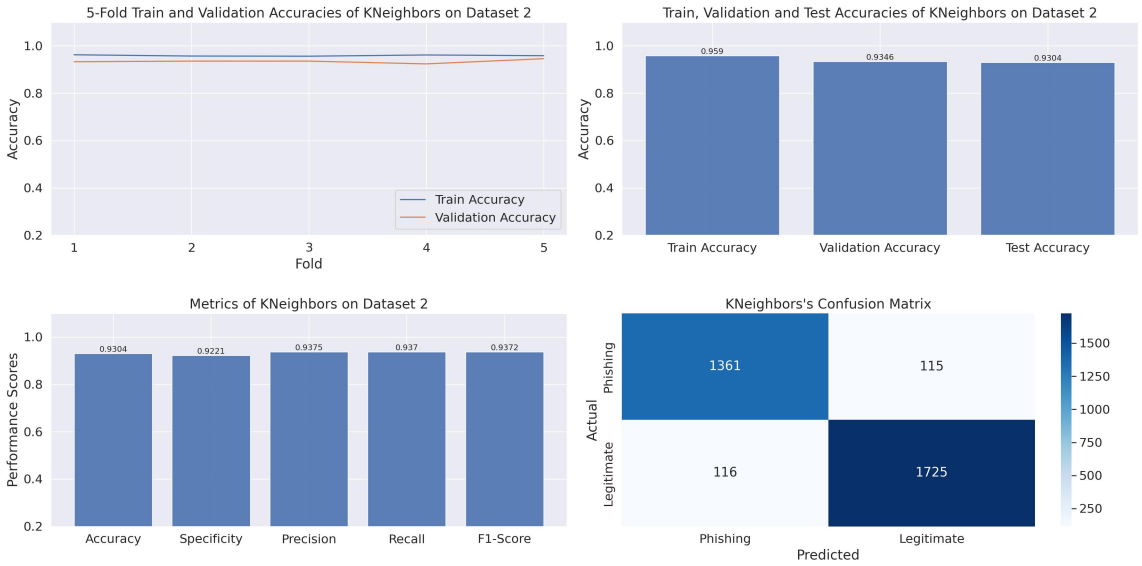

Figure 14: Plot for K-Nearest Neighbors (KNN) on Dataset 2

Plot for K-Nearest Neighbors (KNN) on Dataset 3

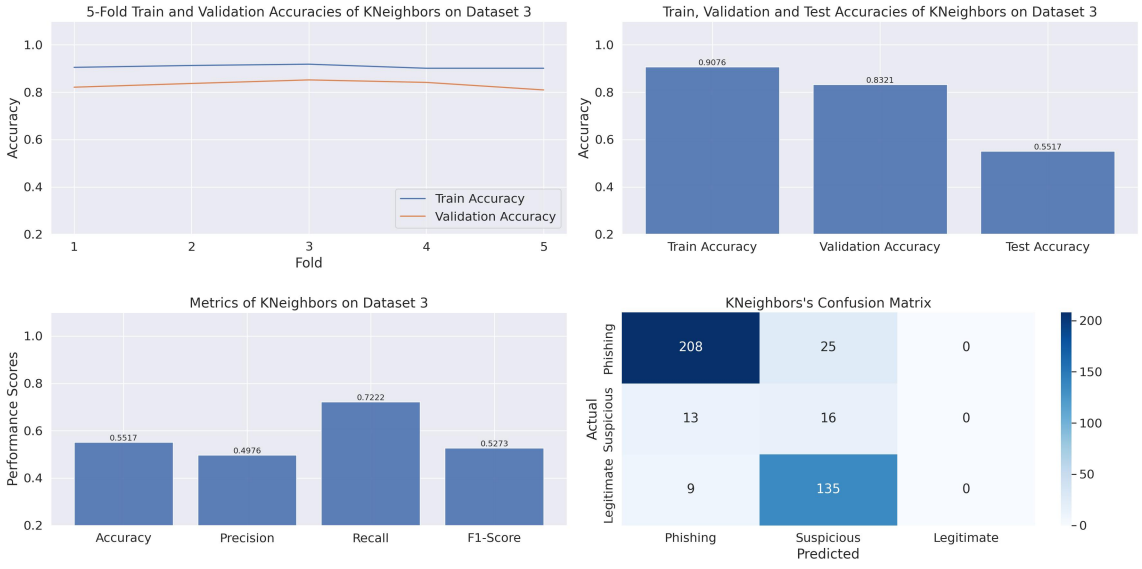

Figure 15: Plot for K-Nearest Neighbors (KNN) on Dataset 3

Plot for Neural Network on Dataset 1

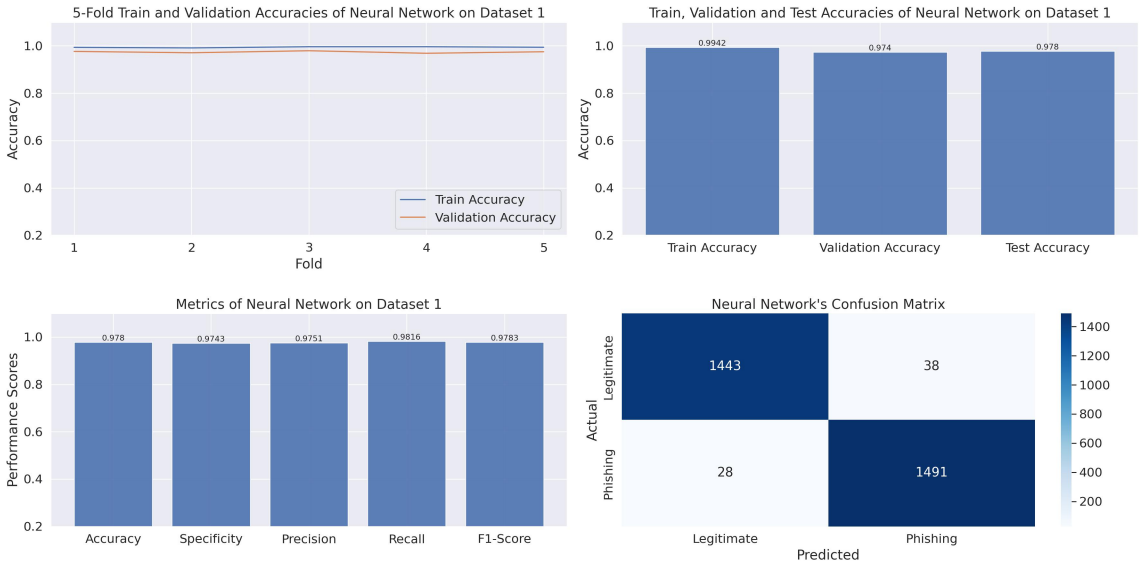

Figure 16: Plot for Neural Network on Dataset 1

Plot for Neural Network on Dataset 2

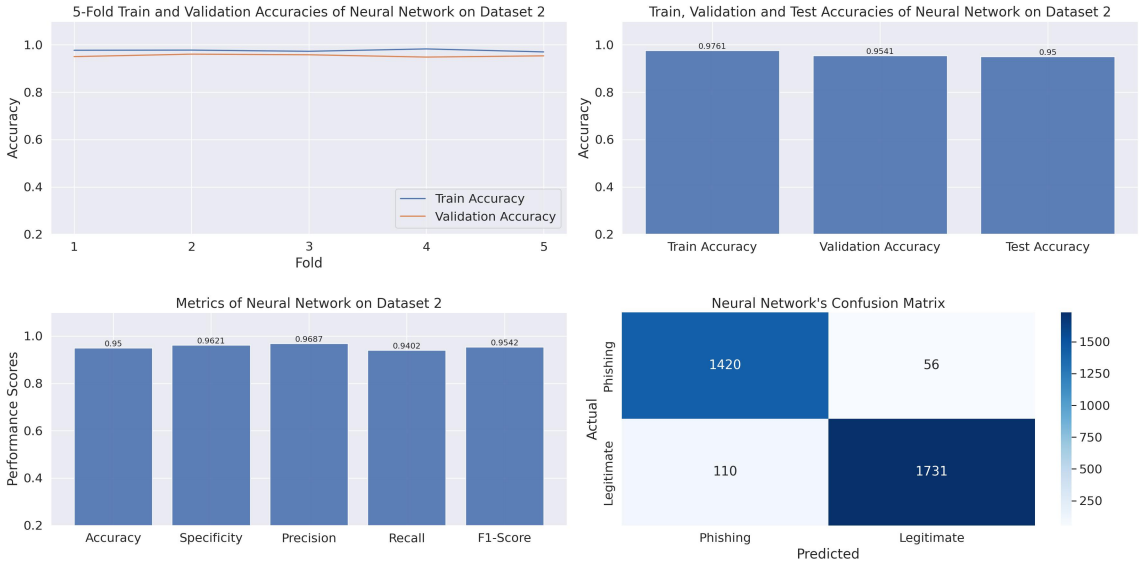

Figure 17: Plot for Neural Network on Dataset 2

Plot for Neural Network on Dataset 3

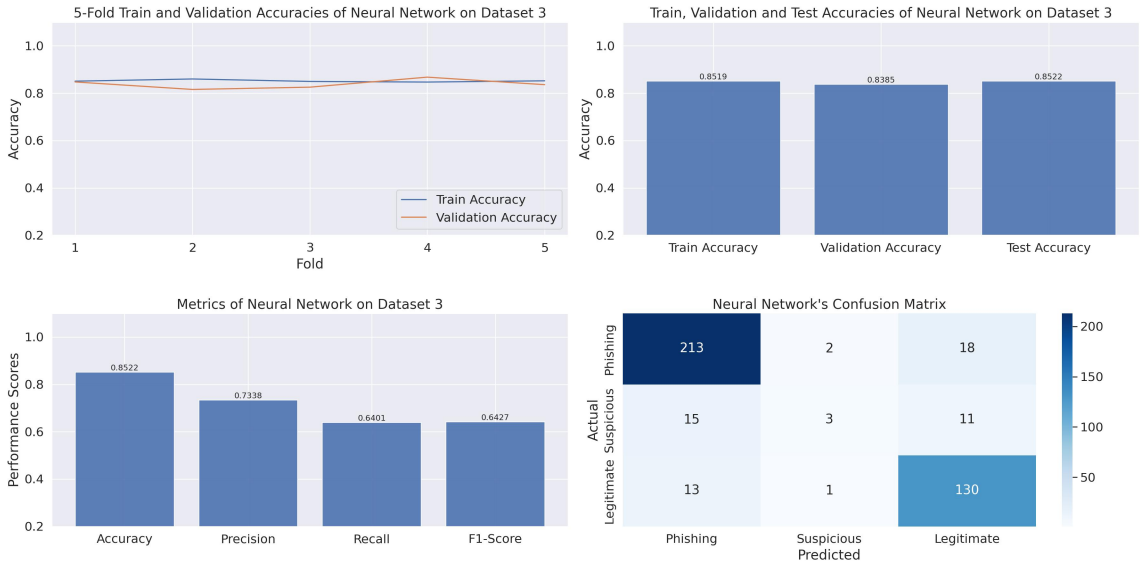

Figure 18: Plot for Neural Network on Dataset 3

Conclusion

Our project is unique compared to other classification systems since it uses a multi-algorithm technique that provides a more customized and efficient method for handling various dataset types. It is so comprehensive, encompassing everything from data preprocessing to a thorough model evaluation. Some competitors may not provide such a comprehensive approach. Utilizing a range of performance measures and integrating cutting-edge machine learning techniques like neural networks demonstrate a depth of research and review that may not be found in all competing solutions.

References

Fatin Almkallawi Abdulhamit Subasi, Esraa Molah and Touseef J. Chaudhery. Intelligent phishing website detection using random forest classifier. In *IEEE 2017 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 2017.

Shouq Alnemari and Majid Alshammari. Detecting phishing domains using machine learning. *Applied Sciences*, 2023.

Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

A.R.Javed G.T.Reddy R.Kaluri G.Srivastava C.Iwendi, Z.Jalil and O.Jo. Keysplitwatermark: Zero watermarking algorithm for software protection against cyber-attacks. *IEEE Access*, 8:72650–72660, 2020.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley Sons, 2012.

Zhile Chen Feng Liu Erzhou Zhu, Yinyin Ju and Xianyong Fang. *DTOF-ANN: An Artificial Neural Network phishing detection model based on Decision Tree and Optimal Features*, volume 95. Science Direct, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science Business Media, 2009.

Tin Kam Ho. The random subspace method for constructing decision forests. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20, pages 832–844. IEEE, 1998.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI*, 14(2):1137–1145, 1995.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Adarsh Mandadi, Saikiran Boppana, Vishnu Ravella, and R Kavitha. Phishing website detection using machine learning. In *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, pages 1–4, 2022. doi: 10.1109/I2CT54291.2022.9824801.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Andrew Kachites McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. *AAAI-98 workshop on learning for text categorization*, 752: 41–48, 1998.

David M Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 2014.

Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Bernhard Schölkopf and Alexander J Smola. Learning with kernels: Support vector machines, regularization, optimization, and beyond. In *MIT Press*, 2002.

Geoffrey John HultenPaul Stephen RehfussRobert RounthwaiteJoshua Theodore Goodman-Gopalakrishnan SeshadrinathanAnthony P. PentaManav MishraRoderic C. DeyoElliott Jeb HaberDavid Aaron Ward Snelling. Finding phishing sites. *Google Patents: Microsoft Corporation*, 2014.

Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

Yasin Sönmez, Türker Tuncer, Hüseyin Gökal, and Engin Avcı. Phishing web sites features classification based on extreme learning machine. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–5, 2018. doi: 10.1109/ISDFS.2018.8355342.

Cornelis Joost Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.