CS361: Phishing URL Detection

Aditya Gupta (210101010) Riya Mittal (210101089) Shally Kandoi (210101096) Shivam Agrawal (210101098)

Abstract

Phishing attacks pose a significant cybersecurity threat by exploiting users to obtain sensitive information through deceptive URLs. This project proposes a novel approach to detect phishing URLs by employing classical machine learning algorithms, such as Support Vector Machines (SVM), Decision Tree, Random Forest and k-Nearest Neighbors. These algorithms are chosen for their efficiency, effectiveness, and the reduced computational resources they require, making them ideal for quick deployment and adaptation to the evolving landscape of phishing tactics.

We will develop a feature extraction framework that targets specific URL characteristics, including domain anomalies, structural peculiarities, and misleading content, to distinguish between phishing and legitimate URLs. A meticulously labeled dataset, combining historical and real-time data, will train and validate our models, ensuring responsiveness to new phishing strategies. The goal is to achieve a high-accuracy detection tool with minimal false positives, suitable for integration into various digital platforms for immediate user protection.

1. Introduction

1.1. Motivation

Phishing attacks are a big problem online, causing financial losses and putting personal information at risk globally. With an average of $136 lost per phishing attack, this amounts to $44.2 million stolen by cyber criminals through phishing attacks in 2021. One of the most expensive phishing attacks was through compromised emails with around 19,369 complaints having a loss of $1.8 billion.

Existing anti-phishing methods commonly rely on either human-verified URL blacklists or employ machine-learning algorithms based on specific features. While human-verified blacklists provide a low false positive rate, they struggle to adapt to new attacks, with studies showing limited effectiveness. Additionally, updating these blacklists demands significant human effort and may lag in responding to emerging phishing threats. Sheng et al. found that blacklists were updated at different speeds, and an estimated 47% - 83% of phish appeared on blacklists 12 hours after they were launched (Sheng et al., 2009). On the other hand, machine learning-based approaches face the challenge of feature quality, often relying solely on URL and HTML DOM, resulting in less discriminative features. This type of method tends to have a relatively higher FP. Concerns over liability for false positives have been a major barrier to deploying these technologies.

1.2. Target Problem

We are dealing with methods for detecting phishing websites by analyzing various features of benign and phishing URLs by machine learning techniques. A phishing URL is a web address designed to impersonate legitimate websites to fraudulently obtain sensitive information, characterized by:

1. Anomalies in domain properties and URL structure.
2. A login form requesting sensitive information such as bank details and password.
3. Content that closely imitates that of legitimate sites but with slight inconsistencies.

2. Data Preprocessing

2.1. Data Collection & Preprocessing

We have used a comprehensive Kaggle dataset comprising of both legitimate and phishing URLs. This dataset originally included approximately 11430 rows and 89 columns, which we eventually reduced to 25 columns.

Data preprocessing refers to the techniques and procedures used to clean, transform, and prepare raw data for analysis. The features are checked for missing or null values, and inconsequential features with only one value across the data or that are redundant are removed.

2.2. Feature Extraction

Different kinds of features are present for classification like URL-based, page-based, domain-based and HTML-based (Mohammad et al., 2012), out of which we have manually extracted all the URL-based features, while the rest are taken from the original dataset.

2.2.1. URL-based Features

We are extracting the following features from the URLs (Xiang et al., 2011):

1. Embedded Domain: Checking whether the URL is similar to a well-known domain.
2. IP Address: Attackers often employ IP addresses in the URL to disguise a webpage's malicious nature, while legitimate websites almost always use domain names instead of IP addresses due to their easy memorability.
3. Number of dots in URL: Phishing pages tend to use more dots in their URLs than legitimate sites.
4. Number of sensitive words in URL: In (Garera et al., 2007), Garera et al. summarized a set of eight sensitive words that frequently appear in phishing URLs. This is a numeric feature with a range of 0 to 8.
5. Out-of-Position Top Level Domain (TLD): Checks for unusual positioning of TLDs in the URL.
6. HTTPS Token: Checks if the website is using HTTPS.
7. URL Length: Phishing websites often have URLs that are much longer than legitimate URLs.
8. Shortening Services in URL: Phishing URLs tend to use URL shortening services to hide the real URL and make it difficult for the user to know the actual URL.
9. Prefix or Suffix Separated by (-): Most phishing websites use a prefix or suffix to the original domain name to make it look like the original website.
10. Number of Special Characters: Phishing websites often use special characters to make the URL look similar to a legitimate website.

2.2.2. Page-based Features

Page-Based Features give information about how reliable a web site is. We have used 2 such Page-Based Features:

1. Global Pagerank
2. Google Index

2.2.3. Domain-based Features

We are extracting the following features from the domain:

1. Domain Registration Length: Phishing websites are often short-lived, and hence have a shorter domain registration length.
2. Domain Age: Phishing websites are often short-lived, and hence have a shorter domain age.
3. Web Traffic: Phishing websites have less web traffic than legitimate websites.
4. DNS Record: Phishing websites often have no DNS records.
5. Whois Registered Domain: Phishing websites often have no whois registered domain.

2.2.4. HTML-based Features

The content of the HTML page can also be used to check if the website is a phishing website. We are extracting the following features from the HTML page:

1. Login Form: Phishing websites often have a login form to steal the user's credentials.
2. Links in Tags: If the website is authentic, we can find at least one hyperlink in the source code of this website.
3. Iframe: Iframes are used to embed another document within the current HTML document. Phishing websites often use iframes to load a different website.
4. Popup Window: Phishing websites often use popup windows to steal the user's credentials.
5. Right Click: Phishing websites often disable the right click to prevent the user from checking the source code of the website.

2.3. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) involves analyzing and visualizing data to understand its characteristics, patterns, and relationships. In our analysis, the features were divided into being categorical or continuous, and both were examined using histograms, box plots, count plots, and heatmaps.

Histograms displayed the distribution of data for continuous features, indicating potential outliers or imbalances. Box plots constructed for these features revealed that legitimate URLs typically have higher page rank or domain age compared to phishing URLs.

Subsequent examination of the distribution of URLs across categorical features using count plots showed limitations in some features, such as 'iframe' and 'popup window', which had minimal variability in URLs. Similarly, highly imbalanced features like 'dns record' provided limited discriminatory power. These plots also highlighted differences in the distribution of features between legitimate and phishing URLs. For instance, legitimate URLs tended to have a higher Google index compared to phishing URLs.

The heatmap shown in Figure 1 also illustrated weak correlations among features, suggesting their independence.

3. Methods Used

3.1. Support Vector Machine(SVM)

We use Support vector machines for the classification of phishing and benign URLs. There are different SVM formulations like C-support vector classification, v-support vector classification, and one-class SVM. We have used **C-support**

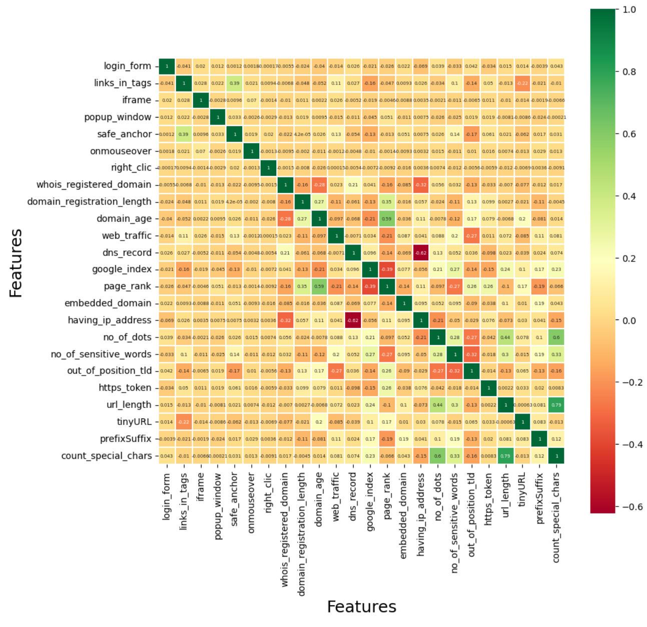


Figure 1. Heatmap

vector classification (Cortes & Vapnik, 1995) because it is ideal for our use case where we have only two classes. Given training vectors $x_i \in \mathbb{R}^n$, $i = 1, \ldots, l$, in two classes, and an indicator vector $y \in \mathbb{R}^l$ such that $y_i \in \{1, -1\}$, C-SVC solves the following primal optimization problem.

$$\begin{aligned} &\min_{w,b,\xi} \tfrac{1}{2} w^T w + C \textstyle\sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \ldots, l \end{aligned}$$

where $\phi(x_i)$ maps x_i into a higher-dimensional space and $C > 0$ is the regularization parameter. Due to the possible high dimensionality of the vector variable w, usually we solve the dual problem.

$$\max_{\alpha} L_{\text{dual}} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$
$$\text{subject to } 0 \leq \alpha_i \leq C, \text{ and } \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0$$
$$\text{where } K(x_i, x_j) \text{ is the kernel function.}$$

3.1.1. Kernel Functions

The results of different kernel functions are summarized in Table 1.

1. Linear Kernel

$$K(x, z) = x^T z$$

2. Polynomial Kernel

$$K_q(x, z) = (c + x^T z)^q$$

We have used $c = 75$ and $q = 2$.

3. Gaussian Kernel

$$K_{\text{rbf}}(x, z) = \exp\left\{-\frac{||x-z||^2}{2\sigma^2}\right\}$$

We have used $\sigma = 1$.

4. Sigmoid Kernel

$$K_{\text{sigmoid}}(x, z) = \tanh(\alpha x^T z + c)$$

We have used $c = 75$ and $\alpha = 0.1$.

5. Laplace Kernel

$$K_1(x, z) = \exp\left(-\frac{\|x-z\|}{\sigma}\right)$$

We have used $\sigma = 1$.

RBF kernel is giving the best results because it works well with non-linear and high-dimensionality datasets similar to ours. The RBF kernel implicitly maps the input features into a higher-dimensional space, where the data becomes more separable. The RBF kernel's smoothness and local nature make it inherently robust to noise in the data. This robustness helps prevent overfitting and allows the model to generalize well to unseen examples.

Kernel function used	Accuracy	LIBSVM Accuracy
Linear	0.87	0.88
Polynomial	0.73	0.91
Gaussian	0.89	0.92
Sigmoid	0.78	0.83
Laplace	0.72	-

Table 1. Comparison of SVM Accuracy with library implementation

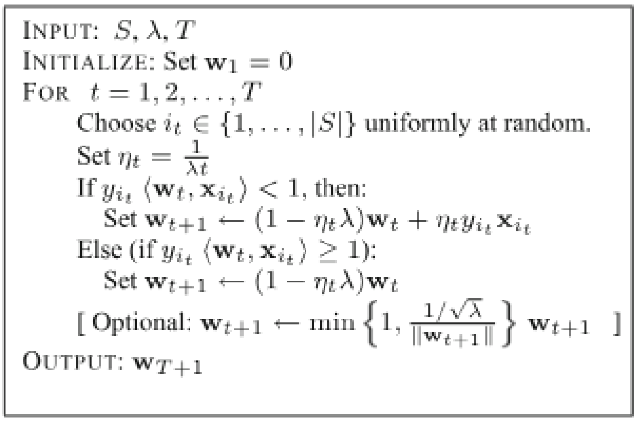
3.1.2. Optimization Algorithms

We have explored various optimization algorithms and analyzed the advantages and disadvantages of each method. The results are summarized in Table 2.

1. **Gradient Descent** The method iteratively updates the parameters of a model by moving in the direction opposite to the gradient of the loss function with respect to those parameters. It is well-suited for convex optimization problems and is thus ideal for maximizing the quadratic objective function of the dual SVM formulation. However, it does not work with large datasets in SVM due to the substantial memory requirements associated with its implementation. To address this challenge, we limit its usage to a training size of 0.4 with our dataset.

2. **Mini-Gradient Descent** Mini-batch Gradient Descent is a variation of the Gradient Descent optimization algorithm. While batch Gradient Descent computes the gradient of the loss function using the entire training dataset, mini-batch Gradient Descent computes the gradient using a smaller subset of the data called a mini-batch. This reduces the memory requirement and allows training SVM with a train size of 0.2 leading to higher accuracy.

3. **PEGASOS with Stochastic Gradient Descent** PEGASOS(Shalev-Shwartz et al., 2007) or Primal Estimated sub-GrAdient SOlver for SVM is a simple and effective iterative algorithm for solving the optimization problem of linear SVM shown in Figure 2. Its rapid convergence properties make it especially well-suited for our application. We achieved an accuracy of 87% using this algorithm and linear kernel due to the ability to train for a larger number of iterations. However, it is important to note that one limitation of PEGASOS is its inapplicability to non-linear kernels.



```
INPUT: $S, \lambda, T$
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
    Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
    Set $\eta_t = \frac{1}{\lambda t}$
    If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
        Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
    Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
        Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$
    [ Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$ ]
OUTPUT: $\mathbf{w}_{T+1}$
```

Figure 2. PEGASOS algorithm

4. **Sequential Minimal Optimization(SMO)** Training a support vector machine requires the solution of a very large quadratic programming (QP) optimization problem. SMO (Platt, 1998) breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets. This allowed us to use a train size of 0.2 without splitting the data into batches. For the standard SVM QP problem, the smallest possible optimization problem involves two Lagrange multipliers, because the Lagrange multipliers must obey a linear equality constraint. At every step, SMO chooses two Lagrange multipliers to jointly optimize and finds the optimal values for these multipliers. Let α_i be the Lagrange multiplier corresponding to the i-th sample, and y_i be its label.
   If the target y_1 does not equal the target y_2, then the following bounds apply to α_2:

$$L = \max(0, \alpha_2 - \alpha_1)$$
$$R = \min(C, C + \alpha_2 - \alpha_1)$$

   If the target y1 equals the target y2, then the following bounds apply to α_2:

$$L = \max(0, \alpha_2 + \alpha_1 - C)$$
$$R = \min(C, C + \alpha_2 + \alpha_1)$$

   The second derivative of the objective function along the diagonal line can be expressed as:

$$\eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2)$$

   We use gradient descent to find the updates in the value of α_2.

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}$$

   where E_i = u_i - y_i is the error on the i-th training example.

   As a next step, the constrained minimum is found by clipping the unconstrained minimum to the ends of the line segment:

$$\alpha_2^{new,clipped} = min(R, max(L, \alpha_2^{new}))$$

   The value of α_1 is computed from the new, clipped, α_2:

$$\alpha_1^{new} = \alpha_1 + y_1 y_2(\alpha_2 - \alpha_2^{new,clipped})$$

5. **Optimized SMO** It achieves roughly the same accuracy as SMO but with faster training rate. SMO chooses the Lagrange multipliers randomly in each round. We can use various heuristics to speed up convergence and we use one such heuristic in this variant of SMO. There are two separate choice heuristics: one for the first Lagrange multiplier and one for the second. The choice of the first heuristic provides the outer loop of the SMO algorithm. The outer loop first iterates over the entire training set, determining whether each example violates the KKT conditions. If an example violates the KKT conditions, it is then eligible for optimization. After one pass through the entire training set, the outer loop iterates over all examples whose Lagrange multipliers are neither 0 nor C The outer loop keeps alternating between single passes over the entire

training set and multiple passes over the non-bound subset until the entire training set obeys the KKT conditions. The second Lagrange multiplier is chosen to maximize the size of the step taken during joint optimization. Based on KKT conditions, stop when: Stop when:

$$\alpha_i = 0 \Rightarrow y_i(w^T x_i + b) \geq 1 - \epsilon$$
$$\alpha_i = C \Rightarrow y_i(w^T x_i + b) \leq 1 + \epsilon$$
$$0 < \alpha_i < C \Rightarrow y_i(w^T x_i + b) \in (1 - \epsilon, 1 + \epsilon)$$

Optimization technique	RBF	Polynomial
Gradient Descent	0.89	0.73
Mini-Batch Gradient Descent	0.89	0.86
SMO	0.90	0.88

Table 2. Comparison of Optimization Algorithms using different kernels

3.2. Decision Trees and Random Forest

We have implemented Decision Trees variants namely: ID3, C4.5, VF C4.5 and CART (Rokach & Maimon, 2005), bagging classifier and Random Forest, from scratch. Results corresponding to different models is shown in Table 4.

3.2.1. Different Variants of Decision Tree

The results for different variants of the decision tree are shown in Figure 3.

1. **ID3 (Iterative Dichotomiser 3)**: is a straightforward decision tree algorithm introduced by Quinlan in 1986. It utilizes information gain as the splitting criterion. ID3 does not handle numeric attributes or (non-categorical input features).

2. **C4.5**: is an advancement of ID3(Mienye et al., 2019), introduced by the same author (Quinlan, 1993). It employs the gain ratio as the splitting criterion. The splitting process ends when the number of instances to be split falls below a certain threshold. C4.5 is capable of handling numeric attributes. We have tried to implement C4.5 using the information gain and information gain ratio and then compared it; we found better results using the information gain ratio for our dataset.

3. **Very Fast C4.5**: is a variant of C4.5 in which mean and median have been utilized as a binary cut-off to estimate threshold values. Once the splitting performances for the mean and median are computed, the cut point that yields the highest value is set as a threshold value for the given attribute. Although this approach has a little less accuracy, directly choosing the mean and median for splitting makes this model train very fast.

4. **CART (Classification and Regression Trees)**: constructs binary trees, where each internal node has exactly two outgoing edges. It uses the Gini Impurity as the criteria for splitting. As our problem is classification, I have implemented only the classification part of the CART.

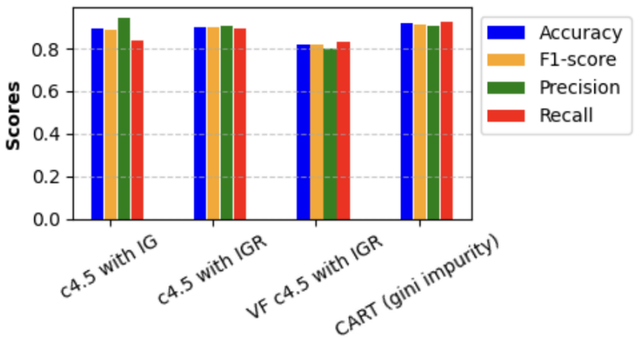


Figure 3. Results for different variants of Decision Trees

3.2.2. Splitting Criterias

1. **Information Gain**: Information gain is an impurity-based criterion that uses the entropy measure (origin from information theory) as the impurity measure (Quinlan, 1987). Formula for InformationGain(a,S) is

$$\text{Entropy}(y, S) - \frac{|\sigma_{a_i = v_{i,j}}(S)|}{|S|} \cdot \text{Entropy}(y, \sigma_a = v(S))$$

where:

$$\text{Entropy}(y, S) = -\sum_{c_j \in \text{dom}(y)} \frac{|\bar{\sigma}_{y=c_j}(\bar{S})|}{|S|} \cdot \log_2 \frac{|\bar{\sigma}_{y=c_j}(\bar{S})|}{|`S|}$$

2. **Information Gain Ratio** : The gain ratio "normalizes" the information gain as follows (Quinlan, 1993):

$$\text{GainRatio}(a_i, S) = \frac{\text{InformationGain}(a_i, S)}{\text{Entropy}(a_i, S)}$$

Note that this ratio is not defined when the denominator is zero. Also, the ratio may tend to favour attributes for which the denominator is very small. Consequently, it is suggested in two stages. First, the information gain is calculated for all attributes. Consequently, considering only attributes that have performed at least as well as the average information gain, the attribute that has obtained the best ratio gain is selected. It has been shown that the gain ratio tends to outperform simple information gain criteria, both from the accuracy and classifier complexity aspects (Quinlan, 1988).

3. **Gini Index**: Gini index is an impurity-based criterion that measures the divergences between the probability distributions of the target attribute's values. The Gini index has been used in various works such as (Breiman et al., 1984) and (Gelfand et al., 1991),

$$\text{Gini}(y, S) = 1 - \sum_{c_j \in \text{dom}(y)} \left(\frac{|\bar{\sigma}_{y=c_j}(S)|}{|S|} \right)^2$$

Consequently, the evaluation criterion for selecting the attribute a_i is defined as: $\text{Gini}(y, S) - \sum_{\bar{\sigma}_{a_i=v_{i,j}}(S)} \frac{|\bar{\sigma}_{a_i=v_{i,j}}(S)| \cdot \text{Gini}(y, \bar{\sigma}_{a_i=v_{i,j}}(S))}{|S|}$ where $v_{i,j} \in \text{dom}(a_i)$.

3.2.3. Ensemble Learning

Applying trees to a data set yields great visualization of the data, but the lack of accuracy and can result in substantial overfitting. This shortcoming was rectified using bagging or RF methods to increase accuracy substantially. The starting point for an ensemble method is a base learner, such as a tree model or a neural network. An ensemble, or collection, of models, is then formed by repeatedly altering the fitting or the training data used by the base learner. The models are averaged (possibly weighted) to form an aggregate estimator.

1. **Bagging Classifier**: The bagging classifier, short for Bootstrap Aggregating, is an ensemble learning method that combines the predictions of multiple base classifiers trained on bootstrap samples of the training data. I have taken each base classifier to be a decision tree.

2. **Random Forest**: Random forests is an extension of bagging. It implements bootstrapping to create a large number of samples. At each node/split, a random sample of predictors is considered to construct the decision tree.

3. **Random Forest vs Bagging Classifier**: The main difference between bagging and random forests is that in random forests, random subsets of predictors are used to build trees. This helps de-correlate trees individually, which reduces the overall variance of the model when averaged. Results for both is shown below in Table 3.

Metric	Bagging Classifier	Random Forest
Accuracy	0.913	0.94
Precision	0.935	0.94
Recall	0.884	0.936
F1 Score	0.909	0.938

Table 3. Performance Metrics of Bagging Classifier and Random Forest

Model	Accuracy	Accuracy
	My Model	sklearn
C4.5		0.908 *(criterion='entropy')*
Inf Gain	0.89	
IG Ratio	0.902	
VF IGR	0.817	
CART	0.917	0.92*(criterion='gini')*
RF	0.94	0.945
Bagging	0.913	0.925

Table 4. Comparison of different models with library implementation

3.3. k Nearest Neighbors (KNN)

In our implementation of the k Nearest Neighbors (KNN) algorithm, we aimed to classify query points based on the majority class among their k nearest neighbors in the feature space.

3.3.1. Classic KNN

The class label of a query point x_q is determined by the majority class among its k nearest neighbors. The class prediction $\hat{y}_q$ for x_q is calculated as follows:

$$\hat{y}_q = \arg\max_y \sum_{i=1}^{k} I(y = y_i)$$

where y_i represents the class labels of the k nearest neighbors of x_q, and $I(\cdot)$ is the indicator function.

3.3.2. Variants of KNN

We implemented and experimented with several variants of the classic KNN algorithm (Uddin et al., 2022) to explore their performance and capabilities. The results are summarized in Table 2.

1. **Fuzzy KNN:** Fuzzy KNN extends classic KNN by considering the degree of membership of each data point to multiple classes. The class membership μ_{ij} of a data point x_i to class j is defined using a fuzzy membership function. The class prediction $\hat{y}_q$ for x_q is computed as a weighted sum of class memberships:

$$\hat{y}_q = \frac{\sum_{i=1}^{k} \mu_{iq} \cdot y_i}{\sum_{i=1}^{k} \mu_{iq}}$$

Key Features:

(a) **Soft Classification:** Unlike the classic KNN algorithm, which makes hard classifications, F-KNN provides soft classifications by assigning membership values to each class.

(b) **Distance-based Weighting:** F-KNN uses distance-based weighting to assign higher weights to closer neighbors and lower weights to farther neighbors when computing class counts.

2. **Weight Adjusted KNN:** The Weight Adjusted KNN algorithm is a variant of the classic K-nearest neighbors (KNN) algorithm that incorporates attribute weighting. In Weight Adjusted KNN, instead of treating all nearest neighbors equally, the algorithm assigns weights to each neighbor based on a kernel function. These weights are used to adjust the influence of each neighbor in the classification process. Common kernel functions include inverse distance weighting, Gaussian kernel, or Epanechnikov kernel. The class prediction $\hat{y}_q$ for x_q is computed as a weighted sum of the class labels of the nearest neighbors:

$$\hat{y}_q = \frac{\sum_{i=1}^{k} w_i \cdot y_i}{\sum_{i=1}^{k} w_i}$$

where w_i represents the weight assigned to the ith neighbor, typically inversely proportional to its distance from x_q.

Key Features:

(a) **Attribute Weighting:** Weight Adjusted KNN assigns weights to neighbors based on a kernel function, allowing it to give more importance to closer neighbors and less importance to farther neighbors.

(b) **Flexibility:** The choice of kernel function provides flexibility in adjusting the influence of neighbors based on distance.

(c) **Robustness to Noise:** By incorporating attribute weighting, Weight Adjusted KNN can reduce the impact of noisy or irrelevant features on the classification process.

3. **Hassanat KNN:** Hassanat KNN incorporates a dynamic neighborhood selection mechanism by adaptively adjusting the value of k based on the local density of data points. The class prediction $\hat{y}_q$ for x_q is determined using an adaptive k-nearest neighbor search within a radius determined by the local data density.

Hassanat Distance Calculation: The Hassanat distance is calculated by finding the maximum and minimum vector points between the two samples and then computing the Euclidean norm of their difference.

Key Features:

(a) **Advanced Distance Metric:** Hassanat Distance KNN uses the Hassanat distance metric, which considers both the maximum and minimum vector points between samples, providing a potentially more informative distance measure compared to Euclidean distance.

(b) **Robustness to Outliers:** By using the Hassanat distance metric, which considers extreme points in the feature space, Hassanat Distance KNN may be more robust to outliers compared to traditional KNN.

4. **Ensemble Approach KNN:** Ensemble Approach KNN combines multiple KNN classifiers to make a final classification decision. This involves techniques such as bagging or boosting, where each base KNN classifier is trained on a subset of the training data or assigned different weights based on their performance. The final prediction $\hat{y}_q$ is obtained by aggregating the predictions of individual KNN classifiers.

Key Features:

(a) **Ensemble Approach:** Ensemble Approach KNN combines predictions from multiple values of k using weighted summation, allowing it to adapt to varying local neighborhood sizes.

(b) **Dynamic Parameter Selection:** By considering a range of k values, Ensemble Approach KNN avoids the need for manually selecting a single fixed value of k, making it more adaptable to different datasets and classification tasks.

Model	Accuracy	Precision	Recall	F1 Score
Classic	0.812	0.777	0.876	0.824
Standard	0.813	0.849	0.763	0.804
Fuzzy	0.839	0.826	0.860	0.843
Weight Adjusted	0.831	0.817	0.855	0.836
Hasannat	0.812	0.778	0.876	0.823
Ensemble	0.826	0.828	0.824	0.826

Table 5. Comparison of Metrics of KNN Variants (for K=4)

4. Results

The results of individual algorithms have been presented in the previous section. Figure 4 compares the results of all the models. From the figure, we can see that Random Forest achieves the highest accuracy and F1 score. Random forests are an ensemble learning method that combines multiple decision trees to make predictions. By averaging the predictions of multiple trees and introducing randomness in the training process, random forests are able to reduce overfitting and improve generalization performance.

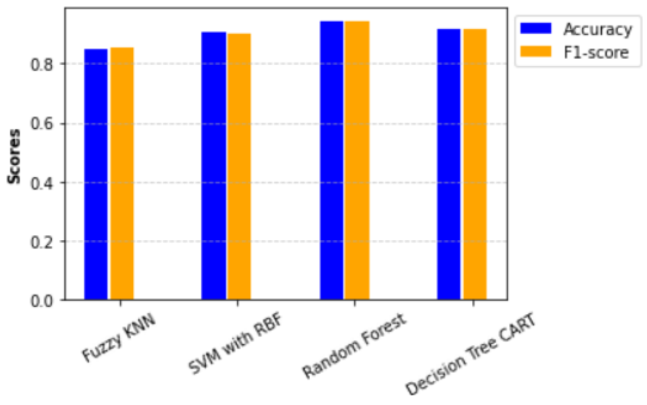


Figure 4. Comparison of different models

5. Future Scope

1. Experiment with additional features or alternative representations of existing features to potentially improve model performance.
2. Investigate the availability of larger and more diverse datasets for phishing URL classification. Look for datasets that cover a wider range of phishing techniques, target industries, and languages to improve the model's generalization ability.
3. Apply Principal Component Analysis (PCA) to reduce the dimensionality of the feature space while preserving the most important information. This can help mitigate the curse of dimensionality, improve model training efficiency, and potentially enhance classification performance.
4. Experiment with genetic algorithms or particle swarm optimization for improved parameter tuning.
5. Explore deep learning techniques such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) for phishing URL classification.

6. Citations and References

References

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Garera, S., Provos, N., Chew, M., and Rubin, A. D. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, pp. 1–8, 2007.

Mienye, I. D., Sun, Y., and Wang, Z. Prediction performance of improved decision tree-based algorithms: a review. *Procedia Manufacturing*, 35:698–703, 2019. ISSN 2351-9789.

Mohammad, R. M., Thabtah, F., and McCluskey, L. An assessment of features related to phishing websites using an automated technique. In *2012 international conference for internet technology and secured transactions*, pp. 492–497. IEEE, 2012.

Platt, J. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

Rokach, L. and Maimon, O. *Decision Trees*, volume 6, pp. 165–192. 01 2005. doi: 10.1007/0-387-25465-X_9.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pp. 807–814, 2007.

Sheng, S., Wardman, B., Warner, G., Cranor, L., Hong, J., and Zhang, C. An empirical analysis of phishing blacklists. 2009.

Uddin, S., Haque, I., Lu, H., Moni, M. A., and Gide, E. Comparative performance analysis of k-nearest neighbour (knn) algorithm and its different variants for disease prediction. *Scientific Reports*, 12(1):6256, 2022.

Xiang, G., Hong, J., Rose, C. P., and Cranor, L. Cantina+ a feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.