Website Spoofing/Phishing Detector (Malicious or Legitimate?)

Justin Bui
Dept. of Computer Science
California State University, Fullerton
Justin_Bui12@csu.fullerton.edu

Nick Sabater
Dept. of Computer Science
California State University, Fullerton
nsabater2022@csu.fullerton.edu

Joshua Gonzalez
Dept. of Computer Science
California State University, Fullerton
JoshuaGonzalez@csu.fullerton.edu

***Abstract*—In this work, our goal is to tackle a real-world problem dealing with the identification of malicious URLs from phishing/spoofing scams and improving the solution for it with innovative ideas by using machine-learning classification. There have been many previous solutions created by companies like Microsoft [4] and Google [5]. There have also been previous machine learning studies to make an effective malicious URL detector [9], [12]. In our approach, we've specifically used K-Nearest Neighbors, Naive Bayes, Decision Trees, and Random Forest classifiers, each trained on full and reduced feature sets of our data. If our machine learning models predict a given URL to be malicious, we will compare the malicious URLs domain name to domain names from a separate data set of legitimate company URLs to identify if the malicious site is disguised as another legitimate entity. After various experiments with our machine learning models, we notice that KNN is the best-performing model when comparing it to other models classifying independently of each other. By amalgamating the ensemble of eight models, comprising four trained on complete feature sets and an additional four trained on reduced feature sets, we achieved the optimal outcome through ensemble voting. The results demonstrated a remarkable accuracy of 0.97, precision of 0.44, recall of 1.0, F1 of 0.61, and AUC of 0.98, highlighting the significant improvements attained by combining the predictions of these diverse models.**

I. Introduction

Ever since the internet has been created, cyberattacks have occurred at an alarming rate and security has become one of the largest, if not, the most important aspect of all time. The internet isn't as safe as people think it is and is filled with malicious websites that could potentially harm users by stealing their personal data, installing malware on their systems, or conducting devastating attacks. It has become challenging for users to identify these types of websites and they often become victims of cybercrime. Our specific goal is to tackle the issue of classifying malicious URLs by creating machine learning models that can detect if a given URL, along with its descriptive features, can potentially serve as a risk to click on. Cybercriminals create malicious URLs as a way to spoof and phish their victims.

Spoofing is when someone disguises an email address, sender name, phone number, or website URL—often just by changing one letter, symbol, or number—to convince you that you are interacting with a trusted source [8]. For example, someone may encounter a link that seemingly contains the domain name of a company, such as "https://www.facebook.com/." In reality, criminals can misspell one letter such as "facebookk," as a trap for victims to click that link.

On the other hand, phishing is a scheme that uses spoofing to lure victims to take the bait. Even with the most keen eye, many can still be at risk of clicking malicious links from phishing attacks. A recent report in 2021 from Cybertalk mentioned that around 30% of phishing emails are opened by unintentionally clicking on malicious links [6]. This motivates us to develop an improved solution by using a machine learning model to perform binary classification on whether URL's are malicious or legitimate given their attributes.

If the URL from our main data set is classified as malicious, we will also identify how it is malicious by comparing it to a separate data set of legitimate company URLs to identify if the malicious site is trying to disguise another legitimate entity by subtly misspelling domain names. This is a method cyber criminals use to phish/spoof their victims.

II. Methodology

This section discusses our initial steps before producing our machine-learning models, which include how and where we collected our data, what we've learned from our data during visualization, and ways to pre-process our raw data into mathematical representations for our machine-learning models to understand.

A. Dataset: Malicious vs Bengin URLs

After evaluating the available options, we found a dataset called "Dataset of Malicious and Benign Webpages," published on Mendeley Data on May 1^{st}, 2020 [1], to be our main data set. This dataset is split into 2 .csv files, each with 1.2 million and 361,934 instances for training and testing data sets respectively. Before this dataset was published online, it was made by scraping millions of URL's around the world using a program called MalCrawler [2]. From there, attributes were extracted from each URL for machine learning applications to classify whether each web page is legitimate or benign. This dataset includes raw page content and JavaScript code, which is useful data to extract further attributes. Malicious or benign

labels made for each URL have been verified using Google's Safe Browsing API [3].

This data set contains 11 attributes:

- url - URL of the web page.
- ip_add - IP Address of the web page.
- geo_loc - The geographic location where the webpage is hosted.
- url_len - The length of URL.
- js_len - Length of JavaScript code on the webpage.
- js_obj_len: Length of obfuscated JavaScript code.
- tld - The Top Level Domain of the webpage.
- who_is - Whether the WHO IS domain information is complete.
- https - Whether the site uses https or http.
- content - The raw webpage content including JavaScript code.
- label - The class label for the benign or malicious webpage (Where "good" = Legitimate URL, "bad" = Risky URL. We will use these terms interchangeably throughout this paper).

B. *Exploratory Data Analysis*

An important step of machine learning is to visualize the training data to gain full insight. We visualize our data with various libraries in Python such as Matplotlib, Seaborn, and Pandas. Initially, we notice this data set does not contain missing values ("NA values"). Fortunately, no imputing nor deleting needs to be done. However, the distribution of labels shows that 27,253 instances are labeled as "bad," while 1,172,747 are labeled as "good," which shows that the data set is skewed. Skewed data can cause issues during classification, such as unequal cost of misclassification errors [9]. In theory, if machine learning classifiers are mostly trained on good data points but not enough bad data points, they will perform well on legitimate URLs, but can mispredict malicious URLs. This can be dangerous as malicious URLs are difficult to overcome, making it crucial to accurately classify harmful links. To address this imbalance, we decide to sample 27,253 good instances out of 1,172,747 to have an equal split between good and bad labels. This method is known as undersampling. Now, the training data has 54,506 instances. This means the testing data set of 361,934 instances is overwhelmingly larger compared to our training set. In this project, we choose to have our training data cover 80% and testing data cover 20% of our entire data. Therefore, we will reduce our testing data by sampling 13,627 instances.

When exploring some categorical features like "https" and "who_is," we observe a separation between good and bad labels. When visualizing the distribution of HTTPS, most malicious URLs are not HTTPS secured while most legitimate URLs are HTTPS secured. This makes sense because websites that do not have secure Hypertext Transmission Protocols enable unencrypted traffic, making it easier for cybercriminals to intercept sensitive information from the user. When exploring the "who_is" attribute, more bad labels have an "incomplete" WHOIS record and more good labels have a "complete" WHOIS record. Most malicious URLs have incomplete WHOIS records because their owners or operators often use privacy protection services to hide their personal information such as name, address, and phone number from public view, as shown in Figure 1. These two attributes will be important to include during machine learning modeling.

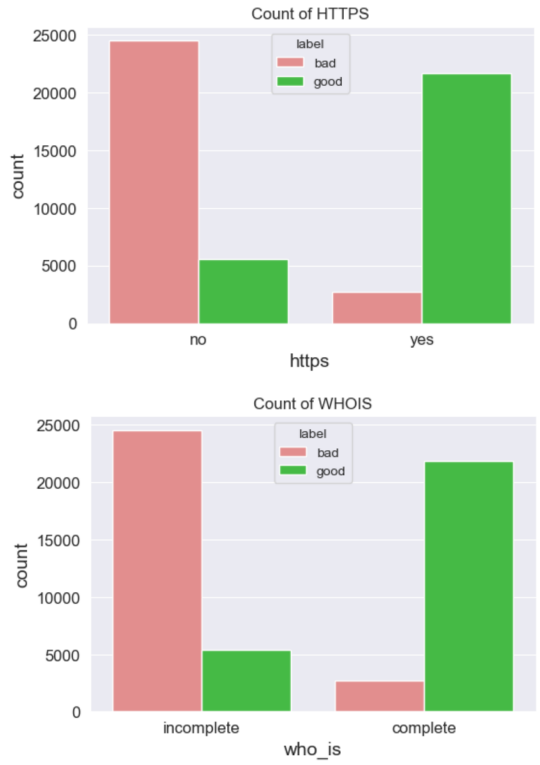

Fig. 1: Distribution of HTTPS and WHO IS Attributes.

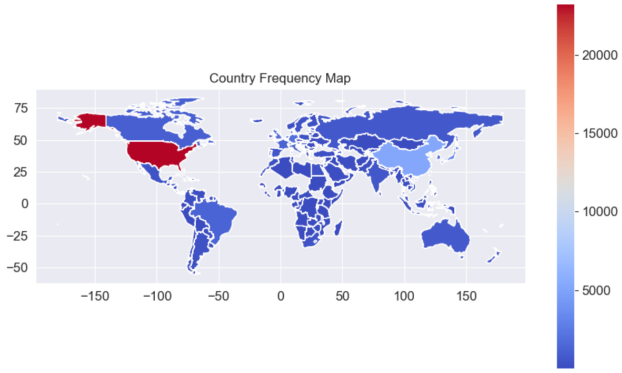

Fig. 2: Spacial data visualization of geographical locations.

We also spatially visualize "geo_loc" using a Python library called Geopandas. Upon graphing the world map, we observe that most of our URLs are located in China and Alaska, as seen on Figure 2. We also attempt to draw the same map based on only "good" data points, and on "bad" data points separately. Regardless, we still see most data points located in the same countries. This shows that there is no

direct relationship between the label and geographical location attributes. Therefore, this attribute will be deleted as it is irrelevant for making predictions.

Next, we explore the "tld" (Top Level Domain) attribute and notice some common TLDs in the training data set include ".com," ".org," ".net," and ".uk". Most TLDs had a fair distribution of good and bad labels. But upon close inspection, we observe that the ".gov" TLD only had good labels, but no bad labels whatsoever. This is an interesting observation, which will be useful during data pre-processing.

Lastly, explore numerical attributes which include "url_len", "js_len" and "js_obf_len" through our summary statistics, as shown in Figure 3. On plotting a Pearson's Correlation matrix in Figure 4, we observe that js_len and js_obf_len have the highest correlation of 0.93, meaning we can consider excluding one of these columns during feature selection.

	url_len	**js_len**	**js_obj_len**
Mean	36.5	332.3	179.5
Std	14.3	267.2	220.3
Min	12	0	0
25%	26	100.5	0
50%	32	190.5	0
75%	44	569.7	361.3
Max	416	854.1	802.9

Fig. 3: Summary Statistics

	url_len	js_len	js_obf_len
url_len	1	0.05	0.05
js_len	0.05	1	0.93
js_obf_len	0.93	0.20	1

Fig. 4: Pearson correlation matrix of numerical attributes.

Now that we've visualized and gained valuable insight into our data, we will now preprocess our raw data into numerical form so that our machine learning models can understand it during training.

C. Data Preprocessing: Numerical Features

Our first step of preprocessing is to handle numerical attributes. As mentioned in the previous section, our numerical attributes are "url_len," "js_len," and "js_obf_len." For these three columns, we will handle outliers and scale them.

We handle outliers using a method called clamp transformation. Clamp transformation is a technique that "clamps all values above an upper and below a lower threshold to the pre-determined upper and lower threshold values" [7]. In this case, we set our upper and lower threshold values based on the upper and lower fence respectively, of our data. Our logic to clamp transformation is seen in the equation represented in Figure 5 below. Suppose, for each value a_i, such that $a_i \in a$, for all i:
In statistics, the upper and lower fences are used to identify outliers. A value a_i is considered an outlier if it is less than the lower fence, or higher than the upper fence. Therefore, a value a_i that is a lower outlier will be assigned as the lower fence, but higher outliers a_i will be assigned as the upper

$$a_i = \begin{cases} \text{lower fence}, & \text{if } a_i < Q1 - (1.5 * IQR) \\ \text{upper fence}, & \text{if } a_i > Q3 + (1.5 * IQR) \end{cases} \quad (1)$$

Fig. 5: Clamp transformation.

fence. We've used this clamp transformation algorithm for all three numerical attributes in this data set. We could use other methods to handle outliers like deleting rows with outliers, but since we want to minimize data loss as best as possible, utilizing transformation techniques like clamp transformation is an effective way to preseve data.

After handling outliers, the next step is to scale "url_len," "js_len," and "js_obf_len." Scaling numerical values is important because, with few exceptions, machine learning algorithms don't perform well when the input numerical attributes have very different scales [11]. We've used a scaling technique called min-max scaling (Also known as normalization), where values in a set of numbers are transformed to be within the range of 0-1. We've normalized each of our columns as seen in the equation represented by Figure 6 below:

$$a_i = \frac{a_i - min(a)}{max(a) - min(a)} \quad (2)$$

Fig. 6: Normalization.

Min-max scaling is effective, as it maintains the shape and distribution of the original data. However, it is sensitive to outliers, as it relies on the minimum and maximum values to squeeze all numbers between 0 and 1. Min-max scaling is the preferred way of scaling our numerical data, as having numbers within this range reduces complexity, thus making it efficient for many machine-learning algorithms to train on.

After preprocessing these three columns, the new scaled columns were renamed as "url_len_scaled," "js_len_scaled," and "js_obf_len_scaled."

D. Data Preprocessing: Categorical Features

The next step of preprocessing is to handle categorical attributes. Again, categorical columns are "tld", "who_is", "https," and "label." Feature encoding was used to transform categorical values into numerical values for our machine-learning algorithms to understand.

Handling "https" and "who_is" was simple as they are binary variables. The "who_is" column contains string values "complete" and "incomplete," so these values will be mapped to 1 and 0 respectively. The "https" column containing string values "yes" and "no" will be mapped to 1 and 0. Lastly, our labels are mapped as 0 for "good" labels (Meaning NOT malicious in) and 1 for "bad" labels (Meaning malicious).

Handling the "tld" was challenging, as this column contains 325 unique TLDs. Our original idea was to use one-hot encoding, which is a useful technique for converting categorical variables with more than 2 categories into a set of binary features. However, since we have an abundance of

unique categories, this may lead to the curse of dimensionality by generating 325 additional columns to our data set. Due to costs like additional memory space to store our data, time complexity during training, and potential hindering of classification metrics, one-hot encoding is not appropriate. However, going back to our step in exploratory data analysis, we observe that the ".gov" was the only TLD that had no "bad" labels. Therefore, we used feature extraction to create a new column called "is_gov_tld" which contains binary values: 1 if the instance is a ".gov" domain and 0 otherwise.

E. Data Preprocessing: Text Features

The final preprocessing step is to handle text attributes. The text attributes in our data set are "url" and "content." To preprocess both text attributes, we utilized an open-source library package in Python called "profanity_check," which provides a simple way to check the presence of profanity in English language text [13]. This library provides a function called $predict\_prob()$, which predicts the probability of a text input containing profanity.

We did this to the "content" column by passing each row of raw text into the function, which is able to parse raw sentences and give a profanity probability. We replaced the "content" column with a new column called "profanity_score_prob," which contains numerical values ranging from 0 to 1.

Since URLs do not have white space, passing a raw URL into $predict\_prob()$ will not give accurate results. Therefore, we must extract only the important keywords of a URL like the root domain and individual names in the path. To do this, we omitted noise in each raw URL such as punctuations, subdomains, and top-level domains. In addition, we lowercase every alphabetical character in the URL. For every URL, each keyword extracted will be put in a string separated by white space. Then, it gets passed into $predict\_prob()$ to give us a numerical score. We then replace the "url" column with a new column called "url_vect," which is a column containing probabilities, similar to "profanity_score_prob."

To provide an example of our final preprocessed dataset, a snippet is presented in Figure 7 below:

	url_vect	is_gov_tld	who_is	https	profanity_score_prob	label	url_len_scaled	js_len_scaled	js_obf_len_scaled
3	0.044	0	0	0	0.901780	1	0.152542	0.842993	0.663632
20	0.005	0	1	1	0.001813	0	0.593220	0.097764	0.000000
64	0.033	0	0	0	0.965517	1	0.864407	0.727081	0.471829
76	0.046	0	1	1	0.049674	0	0.271186	0.038052	0.000000
114	0.046	0	0	0	0.955261	1	0.525424	0.600632	0.421721

Fig. 7: Final preprocessed training data set.

The above prepossessing procedures on our training set will also be performed on our testing set used to evaluate our machine learning models.

III. Machine Learning Models

This section outlines the methodology employed in this research, where we apply various machine-learning techniques to analyze the data. Specifically, we utilize K-Nearest Neighbors (KNN), Naive Bayes, Decision Trees, and Random Forest algorithms. For each of these machine learning algorithms, we trained and tested them based on all features of our data sets. We then had another version where we trained and tested them only on a subset of features after feature selection. We then compare the four classifiers to evaluate malicious and legitimate URLs.

A. Feature Selection: Scikit-Learn

We have our training data set as seen in Figure 7, which contains all 10 descriptive features. We also have another data set where we've selected the top 5 best features using Scikit-Learn's feature selection methods like chi-squared (chi2), ANOVA F-value (f_classif), and mutual information (mutual_info_classif). We found that "who_is," "https," "profanity_score_prob," js_len_scaled," and "js_obf_len_scaled" were the top 5 features that all three methods chose. These features will be our reduced training data set that we also experiment with. A snippet of our reduced version of the training set is shown in Figure 8 below:

	who_is	https	profanity_score_prob	js_len_scaled	js_obf_len_scaled
0	0	0	0.901780	0.842993	0.663632
1	1	1	0.001813	0.097764	0.000000
2	0	0	0.965517	0.727081	0.471829
3	1	1	0.049674	0.038052	0.000000
4	0	0	0.955261	0.600632	0.421721

Fig. 8: Preprocessed training set after feature selection.

B. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a comparison algorithm that compares n neighbors using a distance measure (Normally, euclidean distance) to communally describe the current input data. KNN typically encounters challenges when dealing with large datasets; nevertheless, as we needed to reduce the number of instances from the original dataset and perform undersampling to balance it, KNN fits well within the scope of our project.

Initially, we trained a KNN model with 3 neighbors both on full and reduced feature sets. The obtained results after testing are presented below:

TABLE I: KNN Full Features

Accuracy	Precision	Recall	F1	AUC
0.73	0.07	1.00	0.14	0.86

TABLE II: KNN Reduced Features

Accuracy	Precision	Recall	F1	AUC
0.58	0.05	1.00	0.09	0.79

C. Naive Bayes

Naive Bayes is a stochastic algorithm frequently employed in classification tasks, and it is utilized in this work to classify URLs as either malicious or trustworthy. The algorithm calculates the probability of a hypothesis given observed evidence

(input features), which is proportional to the likelihood of the evidence given the hypothesis. This probabilistic approach enables Naive Bayes to generate reliable classification results based on the available training data. Naive Bayes is based on a famous mathematical theorem known as Bayes's Theorem. In this theorem, the probability of event A given an event B is calculated as the product of the probability of event B given event A and the prior probability of event A, divided by the probability of event B: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$.

In this work, we utilized the Gaussian Naive Bayes [14] from Scikit-Learn, since it is suitable for data with features having a Gaussian distribution. Using default parameters offered by Scikit-Learn's functionalities, our results are shown below:

TABLE III: Gaussian NB Full Features

Accuracy	Precision	Recall	F1	AUC
0.65	0.06	1.00	0.11	0.82

TABLE IV: Gaussian NB Reduced Features

Accuracy	Precision	Recall	F1	AUC
0.77	0.08	0.95	0.15	0.86

D. Decision Trees

Decision Trees are employed to classify URLs by disassembling and converting our data into trees that utilize the significant features and convert them into a straightforward binary decision. Given its simplicity it allows us to scrutinize the data for significant features that could have gone unnoticed due to the uncomplicated nature of the algorithm. However, an issue we need to keep in mind is that it is prone to overfitting.

Using Scikit-Learn, we modeled our own decision tree, initially with our max depth to be at 3. Here are the results for decision tree modeling:

TABLE V: Decision Trees Full Features

Accuracy	Precision	Recall	F1	AUC
0.39	0.03	1.00	0.07	0.69

TABLE VI: Decision Trees Reduced Features

Accuracy	Precision	Recall	F1	AUC
0.39	0.03	1.00	0.07	0.69

E. Random Forests

Random Forest is an improved version of Decision Trees that addresses its tendency to overfit the training data. This algorithm uses bagging, which is an ensemble technique that utilizes multiple different Decision Trees to combine outcomes from random subsets of input, typically through majority voting.

We first set up our model using 100 classifiers (i.e have 100 decision trees within our forest). Our results are shown below:

TABLE VII: Random Forests Full Features

Accuracy	Precision	Recall	F1	AUC
0.39	0.03	1.00	0.07	0.69

TABLE VIII: Random Forests Reduced Features

Accuracy	Precision	Recall	F1	AUC
0.39	0.03	1.00	0.07	0.69

IV. Experiments and Performance Optimization

In this section, we will discuss our experiments to optimize each of the four machine-learning models by tuning hyperparameters for both full and reduced feature sets. Through this experiment, our goal is to find the best combination of hyperparameters and features that can improve accuracy, precision, recall, F1 score, and AUC as possible.

A. KNN: Tuning Number of Neighbors

KNN was optimized using a famous tuning method called the elbow plot to find the best number of neighbors for the model to compare to. Initially, k, our number of neighbors was 3 for each model on full and reduced feature sets. We found the best value of k by plotting many different k values iterating from 10 to 200 on the x-axis and the error rate (Calculated as $1 - Accuracy$) of each corresponding k value on the y-axis of a 2D elbow plot. We graphed plots for both models trained on full and feature sets in Figure 9:

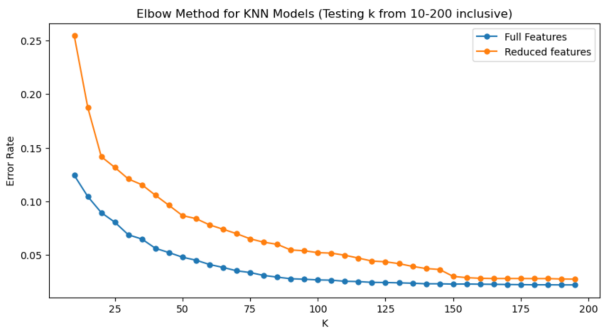

Fig. 9: Elbow plot to find best k-value.

Upon examining the graph presented above, we have encountered a problem. Typically, to find the best k value, we'd want to experiment with even larger k-values to see an increase in our graph in order to identify which k-value is too big, leading to worsened metrics. However, performing an exhaustive search on even larger k values may not be feasible as it would take too much time. Consequently, we opted to restrict the range of our search to values between 10 and 200. As a result, we find that the best k-value for the model with full and reduced feature sets were 190 and 195, respectively.

B. Naive Bayes, Decision Trees, Random Forests

Lots of challenges were faced when tuning for Naive Bayes, decision trees, and random forests, as these classifiers didn't seem to perform well with high-dimensional data, even after

feature reduction, where we chose only the top 5 features out of our entire feature set. Naive Bayes, Decision Trees, and Random Forests are discussed in the same section, as we perform similar experimental procedures for these machine-learning algorithms.

To tune hyperparameters, we use a method called grid search, a brute-force technique to search for optimal hyper-parameters for a given model. Grid search involves coming up with as many possibilities for each hyperparameter, then exhaustively training our model for every combination of hyperparameters until the best performing model is found. We use Scikit-Learn's $GridSearchCV$ function for this [11]. In addition, $GridSearchCV$ also supports cross-validation, which we will be using 3 folds every time this function is used to tune our parameters. At first, we perform grid search on these three models, both trained on full and reduced feature sets. Unfortunately, this process of fine-tuning hyperparameters yields minimal change to our classification metrics. Possible reasons can be:

- The parameters used by default, even before tuning, were already optimal.
- The tested range of hyperparameters do not allow for possible improvements. A wider and more diverse range of parameters could have been explored, but the computational time required was well beyond the scope of this project.

After adjusting the hyperparameters through trial and error, we made the hypothesis that the unsatisfactory outcomes could be attributed to the amount of features used. Of all features in the currently reduced feature sets: "who_is," "https," "profanity_score," "js_len_scaled," and "js_obf_len_scaled," we reduce this feature set even further by choosing just "url_vect," "is_gov_tld," and "js_obf_len_scaled." Upon using a smaller feature set for Naive Bayes, decision trees, and random forests, we notice a significant increase in accuracy. However, precision, recall, F1-score, and AUC scores remained low.

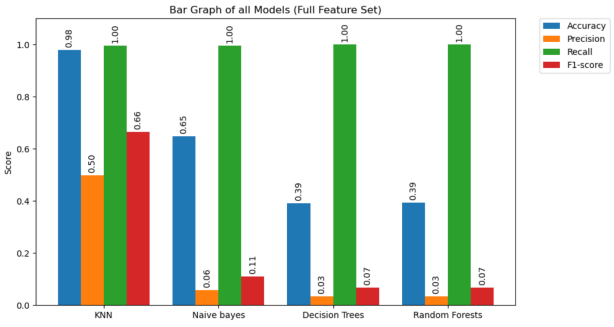

Fig. 10: Models trained on FULL feature sets.

Figures 10 and 11 compare each of our 4 models' performance on full and reduced features after hyperparameter tuning. For all 4 models on full feature sets, we see each has a perfect recall. This is likely due to an imbalanced testing set, as it has much more "good" data points than "bad" data points, therefore reducing the chances of getting false negatives. Of

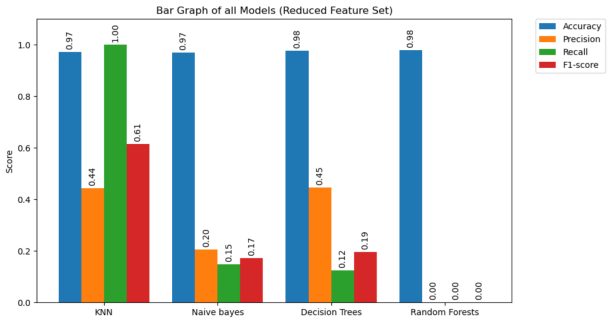

Fig. 11: Models trained on REDUCED feature sets.

all models trained on full features, only KNN has the best accuracy of 0.98, while Naive Bayes, Decision Trees, and Random Forest have accuracy scores less than 0.70.

When tuning each model trained on their respective reduced feature sets, we see a great improvement in accuracy, as each model achieved a score of either 0.97 or 0.98. Unfortunately, reducing feature sets still made no significant improvement on precision, recall, F1 score, and AUC. Notably, random forests exhibit significant deterioration in precision, recall, and F1 scores, with all three metrics registering a value of 0. This is likely because random forest classifiers need an abundance of features to make multiple decision trees (voters), so feeding reducing to just 3 features to train on served no benefit.

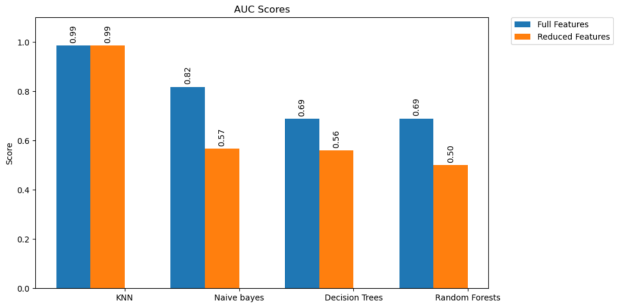

Fig. 12: Final results for AUC scores.

Figure 12 compares AUC scores for all four models separated by full and reduced features. Models trained on full features overall had better AUC scores compared to models trained on reduced feature sets. With the exception of KNN, every other model trained on reduced feature sets seem to have random guessing, as AUC scores were around 0.50. Consistently, we observe that KNN outperforms other models in terms of accuracy, precision, recall, F1, and AUC, whether trained on full or reduced feature sets. This can be due to many reasons, such as its strong ability to distinguish multi-dimensional data using distance measures like Euclidean distance and works flexibly with varying data types.

C. Hard Voting Ensemble

So far, the four models trained on complete and reduced feature sets give subpar results, as we only measured their

performance when classifying independently of each other. We achieved the best final results overall when putting our models together to classify as an ensemble based on majority votes. This is a similar technique used in another study that uses machine learning-based voting with different models for network threat detection in agriculture [10]. This research combined KNN, support vector classification (SVC), decision trees, random forests, and stochastic gradient descent as an ensemble to perform hard voting for their classification problem. We conduct this same process with the four models we made.

We observe variations in model performance across different metrics, with certain models demonstrating superior performance in specific areas while being surpassed by other models in other areas. To further leverage accuracy, precision, recall, F1, and AUC, we place all of our tuned models together as a whole to perform hard majority voting as an ensemble. At first, we created two distinct ensembles: One for models trained on full features, and another for models trained on reduced features.

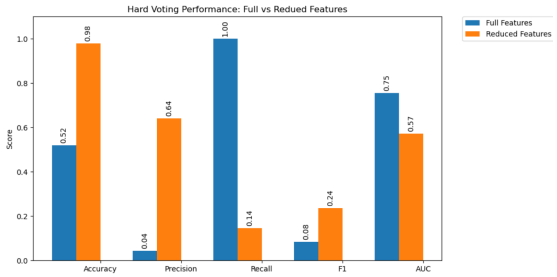

Fig. 13: Ensemble voting on full vs reduced Features

Figure 13 shows performance between accuracy, precision, recall, F1, and AUC for both of these strict voters predicting separately. Still, there are some variations of outstanding and poor metrics. We then combine the four models trained on full feature sets and the other four trained on reduced feature sets, resulting in an ensemble of eight models working collectively to achieve a more balanced outcome of the performance metrics employed in this work. By doing so, the best results were achieved, which are explored in the next section of this paper.

V. Results

Combining the four models trained on full features and the other four on reduced features led to our final results graphed in Figure 14. With some imperfections, like precision and F1 score, we observe an overall increase for all metrics employed in this research. Fortunately, this 8-model ensemble led to a strong accuracy of 0.97, recall of 1.0, and AUC of 0.98.

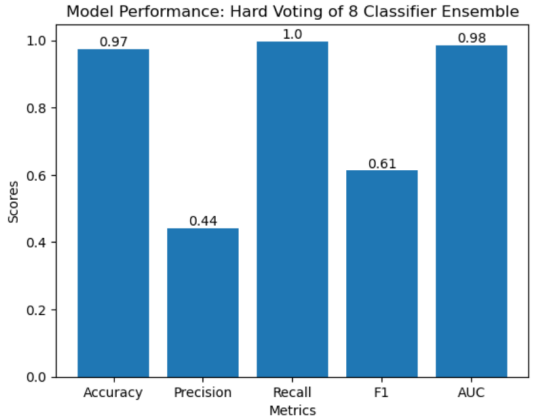

Fig. 14: Ensemble voting on ALL created by far

The improved performance achieved by combining the ensemble of eight machine learning models, four trained on full features and four on reduced features, can be due to several factors. Diversity in feature sets allows each individual model to capture unique patterns and information relevant to the classification task. The models trained on full features can leverage a broader range of input characteristics, potentially identifying complex relationships that may be missed by the reduced feature models. On the other hand, the models trained on reduced features focus on a subset of the most informative attributes, reducing noise and enhancing the models' ability to generalize. By combining these two groups of models, the strengths in some models may overpower weaknesses in others, effectively capturing both detailed and robust feature representations.

VI. Adding Innovation

Apart from classifying malicious and legitimate URLs, our goal is to predict whether a malicious link is disguised as another legitimate entity, famously used for spoofing. To do this, we found another data set, which contains 7+ million legitimate company URLs [15], [16]. Using this data set, we keep only the most relevant features like "name," and "domain."

We use the same ensemble approach to make our prediction of legitimate and malicious URLs. When a data point is predicted as malicious according to majority votes, we retrieve the raw URL from this instance and extract its domain name. To determine if the URL is disguised as a legitimate source, the extracted domain name is compared to the domain names of known legitimate URLs. This comparison is performed using a metric called edit distance, which quantifies the similarity between two strings by measuring the minimum number of operations (insertions, deletions, or substitutions) required to transform one string into the other. By leveraging edit distance, this procedure aims to identify potential criminal activities where malicious URLs attempt to mimic legitimate sources by removing, replacing, or inserting certain characters in the domain name.

	Risky Domain	Shortest Edit Distance	Potentially Disguising As
1	amazon	0	amazon
4	10	1	12
5	buzz	2	bu
9	fca-se	2	case
13	ncc-tu	2	nccu
22	ck	1	bk
37	kempo	2	epo
39	euronet	1	euronext
44	vomit	2	mit
45	home	1	hope
49	hm-le	2	smile

Fig. 15: Edit distance comparisons

We follow this procedure on our testing set. All instances predicted as malicious are stored in a data frame, along with the closest legitimate domain each instance is likely to mock. Figure 15 shows an example of domain names of potentially risky URLs compared with legitimate known URLs using edit distance. We can see a misclassification on index 1 with "Amazon," as this was supposed to be a legitimate URL. However, if the Risky Domain were to be misspelled like "Amason," for instance, this can be a potential risk if the URL is disguised as Amazon's domain name.

VII. Conclusion

In conclusion, we see that KNN has the best-performing metrics on both full and reduced feature sets compared to Naive Bayes, Decision Trees, and Random Forests. When creating our own classifier with all models trained on full feature sets and all models trained on reduced feature sets voting together as an ensemble, we see an overall improvement in accuracy, precision, recall, F1 score, and AUC compared to models predicting independently of each other.

Even though this research has all been finalized, there is still an amount of potential for future work. The mentioned future work encompasses ideas such as performing probabilistic classification. We would be able to determine how much percentage of a website URL is malicious, e.g., based on selected features, this website URL is 75% malicious. If this research were provided an extension of time, it would have been possible to deploy a model to cloud services such as AWS, we could have also constructed a full-stack web application or an API powered by our machine learning models that were utilized.

References

[1] Singh, AK (2020), "Dataset of Malicious and Benign Webpages", Mendeley Data, V1, doi: 10.17632/gdx3pkwp47.1.

[2] Singh, A. K., and Navneet Goyal. "Malcrawler: A crawler for seeking and crawling malicious websites." Distributed Computing and Internet Technology: 13th International Conference, ICDCIT 2017, Bhubaneswar, India, January 13-16, 2017, Proceedings 13. Springer International Publishing, 2017.

[3] Bell, Simon, and Peter Komisarczuk. "An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank." Proceedings of the Australasian Computer Science Week Multiconference. 2020.

[4] Shaikh, Ajaj. "Enhanced Threat Detection with URL Click Alerts by Microsoft Defender for Office 365." Microsoft Tech Community. Microsoft, 21 Mar. 2023. https://techcommunity.microsoft.com/t5/microsoft-defender-for-office/enhanced-threat-detection-with-url-click-alerts-by-microsoft/ba-p/3766110.

[5] Zhu, Bin Benjamin, et al. "US20120158626A1 - Detection and Categorization of Malicious Urls." Google Patents, Google, https://patents.google.com/patent/US20120158626.

[6] "Phishing Attack Statistics 2022." CyberTalk, 30 Mar. 2022, https://www.cybertalk.org/2022/03/30/top-15-phishing-attack-statistics-and-they-might-scare-you/.

[7] Tangirala, Sairam. "Machine Learning for Predictive Analytics: Data Exploration - Sairam Tangirala." Machine Learning For Predictive Analytics: Data Exploration, https://sites.google.com/site/tangiralasairam/personal/machine-learning-for-predictive-analytics.

[8] "Spoofing and Phishing." FBI, FBI, 17 Apr. 2020, https://www.fbi.gov/how-we-can-help-you/safety-resources/scams-and-safety/common-scams-and-crimes/spoofing-and-phishing.

[9] Brownlee, Jason. "Why Is Imbalanced Classification Difficult?" MachineLearningMastery.Com, 14 Jan. 2020, https://machinelearningmastery.com/imbalanced-classification-is-hard/.

[10] Peppes, Nikolaos, et al. "Performance of machine learning-based multi-model voting ensemble methods for network threat detection in agriculture 4.0." Sensors 21.22 (2021): 7475.

[11] Géron, Aurélien. Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed., O'Reilly, 2019.

[12] A. K. Singh and N. Goyal, "A Comparison of Machine Learning Attributes for Detecting Malicious Websites," 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India, 2019, pp. 352-358, doi: 10.1109/COMSNETS.2019.8711133.

[13] Zhou, Victor. "VZHOU842/Profanity-Check: A Fast, Robust Python Library to Check for Offensive Language in Strings." GitHub, 23 May 2020, https://github.com/vzhou842/profanity-check.

[14] Vats, Rohan. "Gaussian Naive Bayes: What You Need to Know?" UpGrad Blog, 22 Feb. 2021, https://www.upgrad.com/blog/gaussian-naive-bayes/.

[15] "B2B Data Provider for Industry Leading Platforms." People Data Labs, www.peopledatalabs.com/.

[16] Labs, People Data. "7+ Million Company Dataset." Kaggle, 10 May 2019, www.kaggle.com/datasets/peopledatalabssf/free-7-million-company-dataset?resource=download.