Machine Learning based COVID-19 Phishing Domains Detection

Paul Kiyambu Mvula
School Of Electrical Engineering and Computer Science
University of Ottawa
Ontario, Canada
pmvul089@uottawa.ca

Abstract*—*Due to the rapid technological advances made over the years, more people are changing their way of living from traditional to more electronic. This transition has attracted and is still attracting cyber-criminals, also called attackers, who make use of the structure of the Internet to commit cybercrimes, such as phishing, to trick users into revealing sensitive data such as personal information, banking and credit cards details, IDs, passwords, and more important information to replicas of legitimate websites of organizations. These cyberattacks have been rising during the COVID-19 pandemic. According to data from the Federal Trade Commission released in September 2020, the loss due to frauds linked to COVID-19 in the US are nearing US$150 million. Because these phishing attacks will only keep increasing, cybersecurity firms and researchers must constantly find efficient ways to tackle this growing issue. Although there are already several anti-phishing approaches in use such as the use of blacklists, visual, heuristics, etc., they cannot efficiently prevent imminent phishing attacks. In this paper, an anti-phishing machine learning model, which uses domain related features to classify COVID-19 related domains between phishing to legitimate, is proposed.

Keywords*—*Machine Learning, Cyber-security, Phishing attacks, Classification Algorithms, Hoeffding Trees, Online learning, supervised learning.

I. INTRODUCTION

The novel corona virus disease or COVID-19 in short, is a contagious respiratory and vascular disease caused by being infected with the Severe Acute Respiratory Syndrome coronavirus 2 (SARS-Cov-2). The virus is thought to be natural from animal origin. The earliest onset of symptoms has been reported by WHO officials on December 8th, 2019 and the outbreak has been declared a pandemic on Mach 11, 2020. It has then infected over 40.5 million people and caused over 1.1 million deaths in 215 countries and territories. Fig. 1 shows the distribution of cases per country as of October 20, 2020 [1].

As a result of the COVID-19 pandemic, public health officials and governments have been working together to lower the transmission and death rate and control the pandemic. They have put in place some measures such as social distancing, mandatory wear of face masks when going in public and crowded places such as supermarkets, frequent hand washing and most importantly they have ordered people to stay at home and avoid unnecessary trips. Some governments imposed stricter measures such as border closures, lockdowns, and curfews.

Despite those measures, the transmission and death toll have not stopped rising. The active

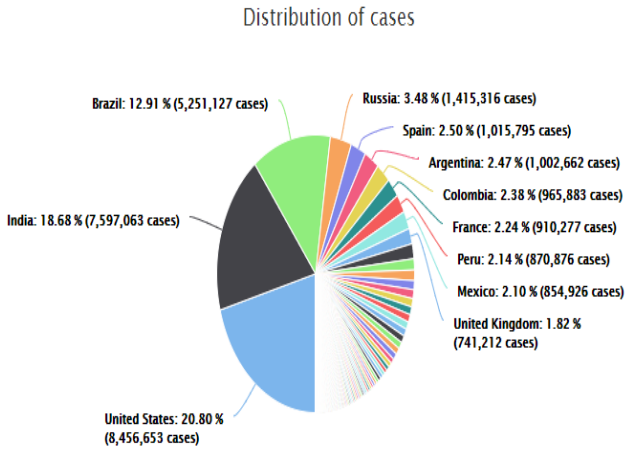

Fig. 1. Distribution of cases per country. Source : https://www.worldometers.info/coronavirus/

measures in place have forced companies worldwide into switching from offline working to remotely working from home, as gathering of more than five people were not allowed by some countries' authorities, and other workers had gone overseas for holidays and were stuck in the countries they were visiting. Schools were also forced to switch to distance learning.

The Anti-Phishing Working Group (APWG) [2], is an international consortium dedicated to promoting research, education, and law enforcement to eliminate online fraud and cybercrime. This year they have already released two Phishing Activity Trend Reports, which analyze phishing attacks and other identity theft techniques, of the first and second quarter of 2020. The number of unique phishing website detected in March has spiked to over 60 thousand, from 40 thousand in February [3]. In Fig. 2, we see the number of phishing sites detected in the first and second quarters of 2020. After the remarkable spike in March, the same month the outbreak was declared a pandemic, the number of phishing sites detected has remained slightly consistent. The most targeted industry sector has remained SaaS (Software-as-a-Service) and WebMail services increasing from 33.5% in the first quarter to 34.7% in the second quarter of 2020. Sales management, customer relationship management (CRM), human resource, billing and other office applications and collaboration tools are components of SaaS. Phishers are interested in stealing SaaS login information because they contain financial and personnel data Anti-phishing developments are being made to prevent phishing attacks by the industry and academia. Le page [4], groups the anti-phishing developments efforts in 3 areas as depicted in Fig. 3.:

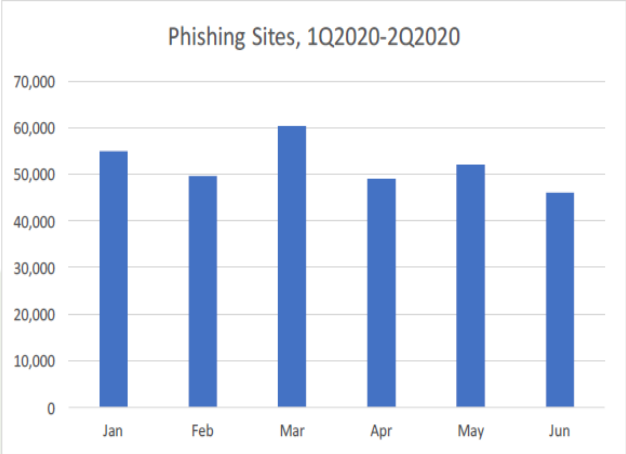

Fig. 2. Phishing sites 1Q2020-2Q2020. Source : http://docs.apwg.org/reports/apwg_trends_report_q2_2020.pdf

- Detection techniques.
- Understanding the phishing eco-system (phishing prevention).
- User Education.

Although user education and understanding the phishing eco-system for phishing prevention are crucial, end-users are always vulnerable to attacks as attackers can use new techniques and exploit those vulnerabilities to trick them into opening links and releasing their personal information. In this paper, we focus more on the phishing detection techniques.

This paper proposes a ML model that classifies domains between phishing and legitimate by using different ML algorithms and compare them to get the best result. We first collect a lot of legitimate and confirmed phishing domains from publicly available data to construct our own data set. After that, we extract useful features from the generated data set and then move on to the model construction and algorithm tuning.

The rest of the paper is organized as follows: in the first following section, related works and literature about phishing domain detection are examined. Section 3 discusses why the detection of phishing domains is difficult. The steps taken in the acquisition of the data set and the feature extraction are explained in Section 4, the experimental setup and proposed model are discussed in Section 5. In Section 6, the results of the different algorithms are presented and compared, and Section 7 contains conclusions and future works on the topic. Finally, section 8 contains acknowledgements followed by references.

II. RELATED WORKS

Software-based phishing detection techniques are mostly divided into three classes: visual similarity-based detection systems [5], list-based detection systems [6, 7] and Machine Learning based detection systems.

A. Visual Similarity based detection systems

Visual similarity-based approaches can be grouped into HTML DOM (Hypertext Markup Language Document Object Model), Cascading Style

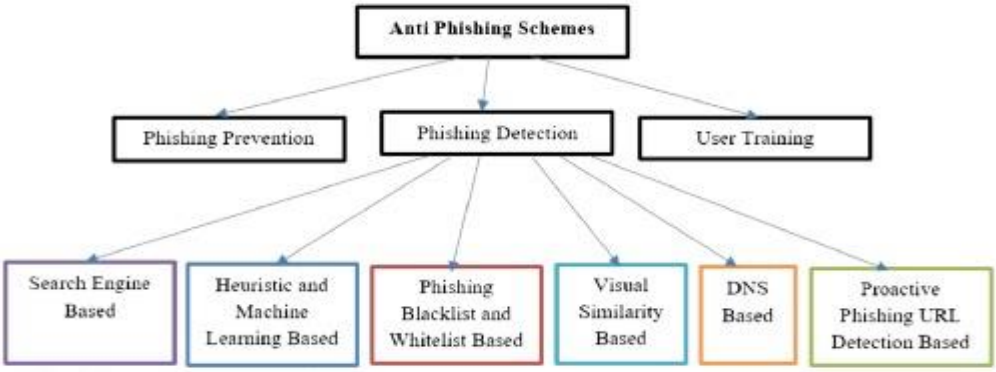

Fig. 3. Anti-phishing Development Areas. Source : https://onlinelibrary.wiley.com/doi/full/10.1002/sec.1674

Sheet (CSS) similarity, visual features, visual perception, and hybrid approaches [9]. Rosiello et al. [12] present an approach based on the reuse of same data across different sites. In the proposed approach, if a client reuses the same data (i.e., same username, password, and so forth) on numerous websites, a warning is generated. The system analyzes the document object model (DOM) tree of the first webpage where the data was at first entered and the other webpage, where the information is reused. If the DOM tree between these two webpages is discovered to be similar, then the system considers it as a phishing attack or else, a legal reuse of data. This system presented high true positive rate of almost 100% but failed when phishing sites contained only images.

Dunlop et al. [13] proposed goldphish, a browser plug-in to recognize phishing sites. It extracts the suspicious site's logo and converts it to text using the Optical Character Recognition software (OCR), the extracted text is then used as a query for the Google search engine. The suspicious site's domain is compared with the top search results, if there is a match, the site is declared as a legitimate, else it is considered phishing. Although it can detect new attacks (known as zero-hour attacks) and identify well-known popular companies' logos, it cannot convert text from logos with dark backgrounds. Chiew et al. [14], proposed an extension of [13] that uses the logo for phishing detection. The approach is divided into two stages: logo extraction and site identity confirmation. This hybrid method uses ML to extract the right site's logo from all pictures present. After that, the right logo is queried in "Google Images" to get the corresponding domains in the results. Then, like goldphish, the suspicious site's domain is compared with the top search results. Like goldphish, it can detect zero-hour phishing attacks; but it has a high false positive rate of 13%. Although these visual similarity-based approaches can achieve high accuracy, their weaknesses are that they are very complex in nature and they have high processing costs because prior knowledge about the websites or large databases of images need to be stored.

B. List-based detection systems

In these systems, there are two types of lists: a list containing malicious domains, the blacklist and another one containing legitimate domains, the whitelist. Blacklists are constructed from previous domains that have been flagged as malicious. These domains can be found in spams, by anti-virus software and from other sources. Once a domain has been blacklisted as a phishing domain, it makes it impossible for the attacker to reuse the same domain to create other URLs from the flagged domain. Although blacklists are updated frequently, black-list based detection systems are not efficient in detecting zero-hour attacks [15]. Even though they have a low false positive rate, their success rate has been estimated to about 20% [16]. This makes black-lists not reliable as a defense system mechanism. Systems in [17] and [18] use an approximate matching algorithm to check whether the suspicious URL can be found in the blacklist or not. Whitelists on the other hand are created by getting specific information about domains from legitimate companies. When an unknown domain appears, the system checks whether it is present in the whitelist, if it is not, it is assumed to be malicious. Cao et al. [7] developed the automated individual whitelist (AIWL), a system that builds a whitelist by registering each site's, visited by the user with a login interface, IP address. When the user will visit a

website, they will be notified if there is inconsistency with the registered information of the visited website. One of the AIWL's pitfalls is that it warns the user each time they visit a site with a login interface for the first time. List-based approaches are limited in a way that the update of those lists in time constitutes a difficult task and often requires a lot of system resources.

C. Machine Learning Based Detection Systems

Machine Learning has proven itself to be a very useful tool not only in cybersecurity but also in other fields of computer science and has been extensively used in literature for phishing detection. In the case of phishing domain detection, it is simply a classification task classifying phishing domains from legitimate to phishing with the generated weights from the features and data set; therefore, to generate a good model, the data set must contain good data and a set of useful features should be extracted from the data.

Jain and Gupta [20] present an anti-phishing approach, that uses machine learning to extract 19 client-side features to classify websites. They used publicly available phishing pages from PhishTank [23] and Openfish [24], and legitimate pages from Alexa well known sites, some top banking websites, online payment gateways, etc. With the use of machine learning, their proposed approach yielded a 99.39% true positive rate. Zhang et al. [10] developed a text-based phishing detection technique named CANTINA, which extracts a feature set from various fields of a webpage using the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm [21]. Then, the top five terms with highest TF-IDF values are queried by Google Search Engine. If the website was included in the top "n" results it was classified as legitimate. However, CANTINA's performance is affected by the language used on the website as it is only sensitive to English. The enhanced model is named CANTINA+ [22], and it includes 15 features taken from different fields of the page like the Document Object Model tree and URL. The system achieved a 92% accuracy rate, but it could produce many false positives. Sahingoz et al. [27] used a data set consisting of 73575 URLs, 37175 phishing URLs from Phish-Tank (2018) [23] and 36400 legitimates URLs collected with Yandex Search API [28]. From these URLs, 40 different NLP based features are extracted with different modules such as the Word Decomposer Module (to separate sub-words of words longer than 7 characters), the Maliciousness Analysis Module (to detect typo-squatting by calculating the Edit Distance between phishing and legitimate URLs) and others, and 1701 word-features are extracted which are then reduced to 102 with a feature reduction mechanism with Weka's built-in functions. The authors obtain an accuracy rating of 97.98% using the Random Forest.

Verma et al. [26] propose a model that separates URLs into n-grams and demonstrate the effectiveness of this method using Shannon's entropy. The authors make a positive contribution by proving that character distributions in the URL's are skewed due to confusion techniques used by attackers. We borrowed the idea of using the Shannon's entropy as a feature from this paper.

In a real-time environment, the detection of a phishing attack should be effective and very fast. List-based approaches are fast, but they are not able to detect zero-hour phishing attacks. On the other hand, visual similarity-based approaches are time consuming, require a lot of memory, and fail to detect zero-hour attacks [25].

III. CHALLENGES IN DOMAIN DETECTION

During the COVID-19 pandemic, attackers have been using several ways to steal information from users and avoid being detected by security systems in place.

A URL or Universal Resource Locator, colloquially termed a web address [8], is a reference to a resource specifying its location on a network. A standard URL begins with the protocol used to access the page. After the protocol, the subdomain and the second Level Domain (SLD) name are found. After the Second Level Domain (SLD) name comes the Top-Level Domain (TLD) name. Following is the path to the requested file and finally the filename. Fig. 4. shows the standard URL structure, when grouped the SLD and TLD are referred to as Domain name, which constitute the most important part of a URL as the SLD can only be set once, which is upon creation.

This paper focuses on the detection of phishing domains by looking at:

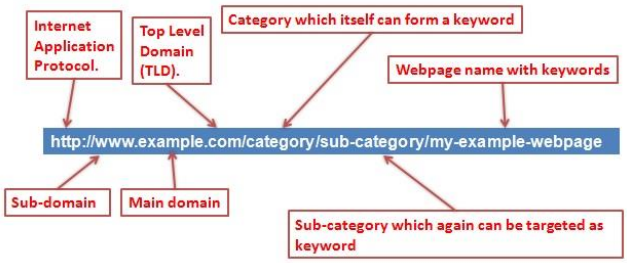

Fig. 4. Standard Structure of a URL. Source : https://www.kvrwebtech.com/blog/seo-friendly-url-structure-mystery-resolved/

- The Sub-Domain.
- The Second-Level domain (SLD) name.
- And the Top-Level Domain (TLD) name.

To increase the attack's performance and get more victims, attackers mainly use a combination of techniques such as random characters, typo-squatting, etc., in the domain name to trick the user into believing they are browsing a legitimate website.

Because we will only use publicly available information to detect phishing domains, regulations such as the General Data Protection Regulation (GDPR), which prevents registration information to be made available to the public, set up walls between researchers and useful registration information that can be used in phishing detection techniques. This, in turn, makes it difficult to come by a high quality worldwide accepted data set entirely consisting of verified phishing and legitimate domains or at least with low false positives and false negatives. The steps taken in the obtention of the data set and the extracted features are described in the next section.

IV. DATA SET AND DATA PREPROCESSING

In computer science, the quality of the output is determined by the quality of the input, as stated by George Fuechsel in the concept "Garbage in, Garbage out.". Therefore, a not only a good data set was needed for the implementation and comparison of the models, but also a set of useful features were extracted from the data set.

1. *Data set*

We constructed our own labeled data set containing two classes of domains: legitimate and phishing domains. The phishing domains came mainly from DomainTools [35] and PhishLabs [36]. DomainTools provides a free, publicly available list of COVID-19 related domains updated daily. On their list, each domain is associated with a risk score of 70 or higher by DomainsTools' risk scoring mechanisms. For this project, we only kept domains with a risk score greater than or equal to 90 because these domains are not manually checked and the higher the risk score, the more likely the domains were to be confirmed phishing. PhishLabs on the other hand, provided intelligence on COVID-19 related attacks from March to September 2020, and the information collected on malicious domains is available on their website.

We tried to find a publicly available list of confirmed legitimate domains accepted worldwide, however, we could not. Therefore, we compiled our own list with the use of search engines APIs. We used Google Custom Search [19], Bing Search [38], Yandex Search [28] and MagicBaidu Search API [37]. First, a specific keyword_list was built with COVID-19 related keywords such as "corona, covid, wuhanvirus, novelvirus, …", and afterward these words were sent to the search APIs to get the first hundred results, which had a very low chance to be phishing pages. This originates from the fact that; web crawlers do not give high rank to the phishing domains because of their short lifetime.

After these efforts, our final data set consisted 67616 domain names, of 5971 legitimate domains from the search APIs and 61645 malicious domains collected from DomainsTools and PhishLabs. We then constructed an unlabeled data set to make predictions for with domains from ZETAlytics, the unlabeled data set consists of over 350000 domain names.

2. *Feature extraction*

After building the data set, extracting a set of good features was the next important step. We extracted a set of lexical features from the domains in the data set. Initially, 12 features, grouped in 9, were extracted:

1. ***Length***:

   i. *Number of Words (numeric):* this feature checks the number of words in the domain name.

   ii. *Number of Characters (numeric): this feature* checks the number of characters in the domain name.

2. ***Containing "-"*** *(numeric):* this feature checks whether there is a hyphen in the domain name.

3. ***Entropy of domain*** *(numeric):* this feature calculates the Shannon Entropy of the domain name. The Shannon's entropy allows to estimate the average minimum number of bits needed to encode a string of symbols based on the alphabet size and the frequency of the symbols as shown in (1), where $p(x_i)$ divides the number of appearances of a character in the string by the length of the string.

$$H(x) = -\sum_{i=0}^{n} p(x_i) \log_b p(x_i) \quad (1)$$

We calculated three entropy values:

i. The entropy of the domain name with the subdomain and suffix

ii. The entropy of the domain name and the subdomain without the suffix

iii. The entropy of the domain name without the subdomain and suffix

4. ***Tranco Rank*** (numeric): this feature checks whether a domain is on the Tranco [39] top 1 million list of domains. Domains on this list are most likely to be legitimate ones.

5. ***Ratio of the longest word*** *(numeric):* this feature matches the longest word that a domain name contains and normalizes it by dividing by the length of the domain name.

6. ***Typo-squatting*** *(numeric):* this feature checks whether a domain name contains typos by generating a list of fuzzy domains from a given list of keywords with bitsquatting, addition/insertion/repetition/hyphenation/replacement, transposition, vowel swaps, homoglyphs, … making use of dnstwist [40], a domain name permutation engine for detecting homograph phishing attacks, typo squatting, and brand impersonation.

7. ***Freenom TLD*** *(numeric):* this feature checks whether the top-level domain belongs to Freenom [41], which provides free domain names on ".gq", ".cf", ".ml", ".tk" and ".ga". Therefore, most of these domains are used for malicious intent.

8. ***Numbers other than 19*** *(numeric):* this feature checks whether a domain contains number other than 19. Hao et al. [42] observe that spam and phishing domains are more likely to use numerical characters than legitimate domains. Possible reasons may be that attackers add digits to generate several names from the same word or generate random names from a character set containing digits. We did not check only if the domain contained numbers because our data set consists of "COVID-19" related domains, therefore most the domains, if not all, contained "19". That is why we were more interested in seeing whether a domain name contained other numbers.

9. ***Number of subdomain levels*** *(numeric)*: This feature counts the number of subdomains in domain. For example, if a domain name is "a19.b18.example.com" the number of known subdomains for "example.com" is two: "a19" and "a18". As for the previous feature, this domain also contains a number other than "19", which is "18".

10. ***Label*** *(numeric)*: the label of the domain, 1 for phishing and 0 for legitimate.

Except from the Tranco Rank, other web-based features such as the data of registration and age of the domain were not used in this project because the amount of time required to fetch them, and it could be days for a huge data set.

Because our data set is highly imbalanced (90% more phishing domains than legitimate), we applied the NearMiss [45] under-sampling algorithm to balance the data set to a ratio of 20:80. NearMiss is a near-neighbor under-sampling technique that balances class distribution by randomly eliminating majority class examples. When instances of two different classes are very close to each other, the instances of the majority class are removed to

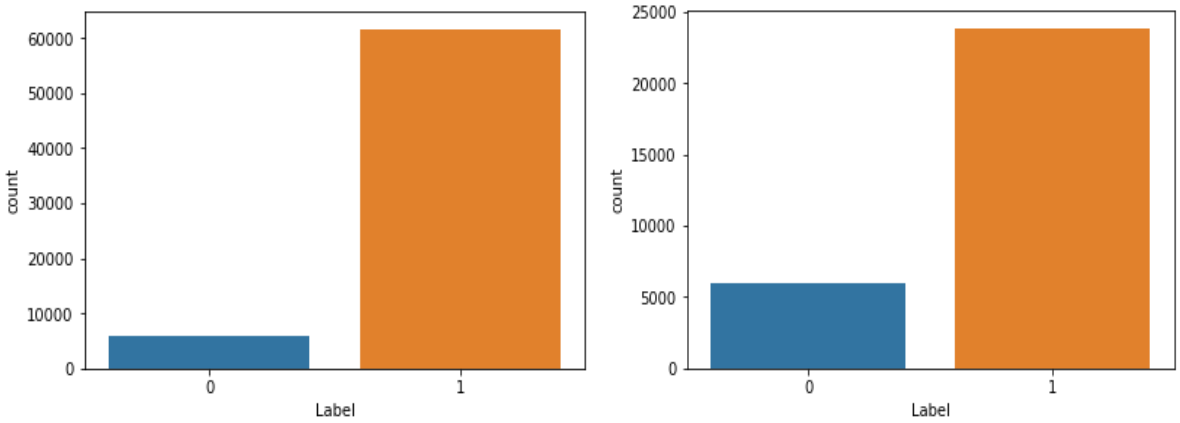

Fig. 6. Data distribution before (left) and after (right) NearMiss under-sampling algorithm.

increase the spaces between the two classes to help in the classification process. Fig. 6 shows the data distribution before and after under-sampling the majority (phishing) class. Class 0 constitutes the legitimate domains and class 1, the phishing domains.

V. EXPERIMENTAL SETUP

We applied two architectures to our data set: lambda and kappa architecture. In the lambda architecture, all the data is already available to us therefore, the model is treated as a static object: it is trained on the available data set and then makes predictions on new data. To learn from new data, the model should be retrained from scratch. This is referred to as batch-learning in literature. In the kappa architecture on the other hand, the model can learn as instances arrive, this is referred to as online learning. It differs from batch learning in a way that the online model, in addition to making predictions for new data, is also able to learn from it, we do not have a fixed data set, but a data stream that is continuous and temporary ordered. Fig. 7 shows the workflows of the different architectures.

In the lambda architecture, we applied 6 machine learning algorithms: the Decision Tree Classifier (DT), the Random Forest Classifier (RF), the Gradient Boosting Classifier (GBM), the Extreme Gradient Boosting (XGB) [29], the Support Vector Classifier (SVM) and the Multilayer Perceptron (MLP). Except from the Extreme Gradient Boosting algorithm which has been developed in C++ and downloaded separately to use in the Python Jupyter NoteBook, all the algorithms have been implemented the scikit-learn library [11] in python.

In the kappa architecture, we applied one algorithm, the Very Fast Decision Tree (VFDT), or Hoeffding Tree classifier (HT) [46]. The authors state that to find the best attribute to test at a given node, it may be sufficient to consider only a small subset of the training examples that pass through that node. Thus, for a given stream of examples, the first ones are used to choose the root test; and once the root attribute is chosen, the succeeding examples are passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on recursively.

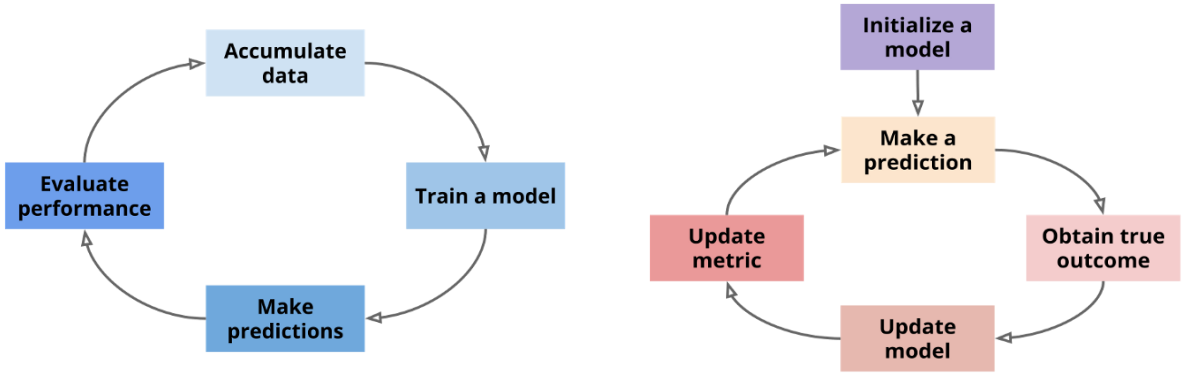

Fig. 7. Lambda (left) and Kappa (right) architectures. Source : https://maxhalford.github.io/slides/the-benefits-of-online-learning/#1

Hoeffding trees solve the difficult problem of deciding exactly how many examples are necessary at each node by using the Hoeffding bound (or additive Chernoff bound) [47, 49]. Consider a real-valued random variable r whose range is R (e.g., for a probability the range is one, and for an information gain the range is $\log c$, where c is the number of classes). Suppose n independent observations of this variable have been made and their mean is $\bar{r}$. The Hoeffding bound states that, with probability $1 - \delta$, the true mean of the variable is at least $\bar{r} - \varepsilon$, where:

$$\varepsilon = \sqrt{R^2 \ln \left(\frac{1}{\delta}\right) / 2n} \quad (3)$$

This shows that the Hoeffding bound is independent of the probability distribution generating the observations. Not only does it achieve high accuracy with a small sample, but it is also incremental and does not perform multiple scans on the same data. We used the Hoeffding Tree classifier developed in scikit-multiflow [53], the machine learning package for streaming data. The major draw back of the HT is that it cannot handle changes in the data distribution i.e., concept drift.

The Decision Tree Classifier is among the popular and powerful algorithms used for supervised learning. It helps select appropriate features for splitting the tree into subparts and finally getting the target item. At the bottom of the tree, each leaf is assigned to a class. It can be used for classification as well as regression. The attributes of the samples are assigned to each node, and each branch's value corresponds to the attributes [30].

The Random Forests are ensemble algorithms consisting of several Decision Trees which are used as voters. The term "ensemble" refers to the grouping of two or more weak learners or weak learning algorithms to improve the performance of the performance of a single classifier [31]. The algorithm was developed by Breiman and Cutler [32]. Random forests attain high accuracies and can handle outliers and noise in the data. In RF, each tree is trained with a random sub-sample drawn from the main training set. Random Forests are powerful in a sense that because the decision comes from the aggregate of all the decisions of the trees in forest, they are less prone to over-fitting and their accuracy is relatively high. RF runtimes are quite fast, and they can deal with unbalanced and missing data. Their weaknesses are that when used for regression they cannot predict beyond the range in the training data, and that they may overfit data sets that are particularly noisy [50]. More details about the Random Forest can be found at [33].

The Gradient Boosting Classifier [43][44] is also an ensemble method which uses decision trees as weak learners, but is a combination boosting with elements from Deep Learning and Neural networks where instead of having a formula to calculate the weights, a gradient descent is used to focus on examples that are difficult to learn. The objective of Gradient Boosting is to minimize the loss function, or the difference between the actual class value of the training example and the predicted class value, using 1st order derivatives.

The Extreme Gradient Boosting [29] is also a boosting method that builds decision trees sequentially. Previous weights are fed to the next decision tree until a termination criterium is met. XGBoost is a refined and customized version of a gradient boosting decision tree system, created for performance and speed. The term "eXtreme" refers to the fact that the algorithms and methods have been customized to push the limit of what is possible for gradient boosting algorithms. Fig. 5 shows the workflow of the extreme gradient boosting algorithm. It can be used for both classification and regression as well. Among the algorithm's features, the regularized boosting which prevents overfitting, the ability to handle missing values automatically and the tree pruning which results in optimized trees are what make the algorithm very powerful.

Gradient boosting and Extreme Gradient Boosting's weakness is that they are more likely to overfit the data due to those decision boundaries, but they are strong in achieving high accuracies, especially the XGB.

The Support Vector Classifier is also an important algorithm used in Machine Learning. It has been proposed by Vapnik [34]. It is used for both classification and regression. It is a geometric model that finds hyperplanes to create boundaries between classes. By using kernel functions and the kernel trick, which calculates the high dimensional relationships without actually transforming the data in a higher dimension, it reduces the amount of computation and makes the infinite dimensions used by kernel functions such as the

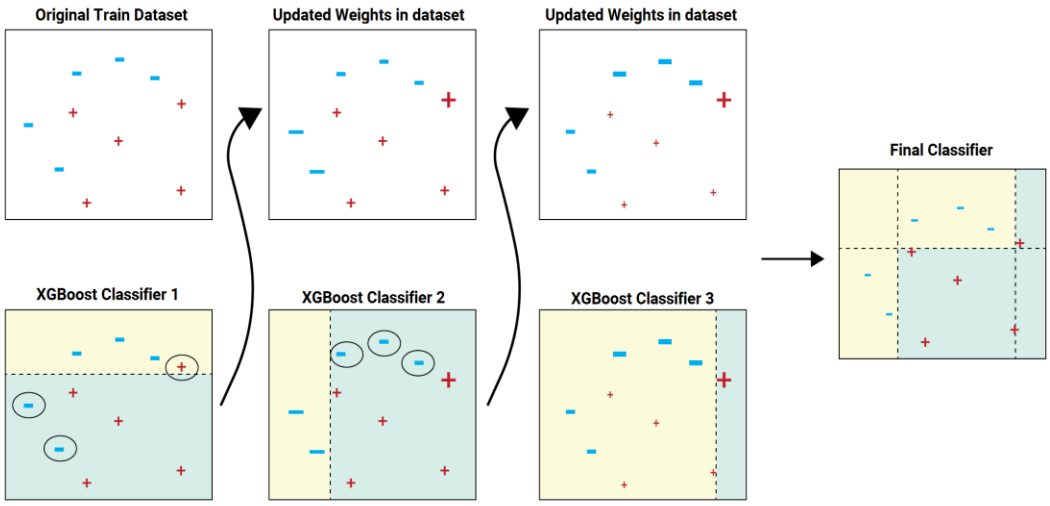

Fig. 5. eXtreme Gradient Boosting workflow. Source : https://blog.quantinsti.com/xgboost-python/

Radial Kernel, which states that the similarity between two points in the transformed feature space is an exponentially decaying function of the distance between the vectors and the original input space as shown in (2), possible.

$$K(x, x') = \exp(-\gamma ||x - x'||^2) \quad (2)$$

The multilayer perceptron (MLP) is a class of feedforward artificial neural network. It consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function to map the weighted inputs to the output, two common activation functions are sigmoid, shown in (4-5). MLP uses the back propagation supervised learning technique for training [51-52], a generalization of the least mean squares algorithm in the linear perceptron.

The degree of error in an output node j at the n^{th} sample is represented by $e_j(n) = d_j(n) - y_j(n)$, where d is the target value and y is the value predicted by the perceptron. The node's weights are then adjusted based on corrections minimizing the error of the output, given by (6), then using gradient descent, the change in weight $\Delta w_{ji}(n)$ is defined by (7), where y_i is the output of the previous neuron and η is the learning rate selected to ensure a quick convergence of the weights to a desired response without oscillations.

$$y(v_i) = \tanh(v_i) \quad (4)$$

$$y(v_i) = \frac{1}{1+e^{-v_i}} \quad (5)$$

$$\varepsilon(n) = {}^1/_2 \sum_j e_j^2(n) \quad (6)$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial v_j(n)} y_i(n) \quad (7)$$

Neural networks can learn the function mapping the input to the output without it being explicitly provided and can also handle noisy data and missing values. However, their key disadvantages are that the answer that emerges from a neural network's weights is difficult to understand, it is sort of a black box, and the training can take longer than certain other methods of machine learning, such as the ones mentioned earlier.

VI. Experimental Results

For the supervised learning approach, the train and test have been split with scikit-learn built-in function. the train set consists of 66.66% of the total number of records under-sampled with NearMiss, and the test set consists of 33.33%, after testing, we moved to predicting classes for our unlabeled data set. For the online learning approach on the other hand, we trained the HT classifier with our entire labeled data set and moved to make predictions for our unlabeled data set.

Experiments were carried out on a Hewlett Packard device with 2.6 GHz Intel Core i7 processor and

TABLE. II. DT 10-fold cross-validation

Fold	Time	F-1	PRECISION	ACC.	TPR	TNR
1	0.157908	0.868091	0.974453	0.809779	0.782663	0.918060
2	0.135905	0.956207	0.971490	0.931011	0.941398	0.889447
3	0.142918	0.958396	0.977372	0.934695	0.940142	0.912898
4	0.141918	0.976744	0.977768	0.962827	0.975722	0.911223
5	0.139908	0.974564	0.978885	0.959478	0.970280	0.916248
6	0.155908	0.976968	0.976968	0.963149	0.976968	0.907873
7	0.135921	0.971645	0.974716	0.954774	0.968593	0.899497
8	0.134942	0.973325	0.976401	0.957454	0.970268	0.906198
9	0.137920	0.974628	0.976060	0.959464	0.973199	0.904523
10	0.159891	0.965459	0.971186	0.945059	0.959799	0.886097
AVG	0.144314	0.959603	0.975530	0.937769	0.945903	0.905206
STD	0.009267	0.031305	0.002445	0.044023	0.055853	0.010184

8 GB of 1600MHz DDR3 RAM. Each test is executed with the different algorithms trained with the corresponding training data set.

Frequently, to evaluate the performance of a model, a confusion matrix is constructed and from the values of the confusion matrix, metrics such as the accuracy, the f1 score, the specificity (True Negative rate, recall of the negative class), the sensitivity (True Positive Rate, recall of the positive class), precision…, can be calculated to evaluate the efficiency of an algorithms. Equations to calculate the metrics are shown in (8-12). Where TP consists of the true positives, the domains predicted as malicious that are truly malicious, TN the true negatives, the domains predicted legitimate that were truly legitimate, FP the false positives, the domains predicted as malicious but are in fact legitimate and FN the false negatives, the domains predicted as legitimate but are in fact malicious, produced by the algorithms.

$$Sensisivity\ (TPR) = \frac{TP}{TP\ +\ FN} \tag{8}$$

$$Specificity\ (TNR) = \frac{TN}{TN\ +\ FP} \tag{9}$$

$$Precision = \frac{TP}{TP\ +\ FP} \tag{10}$$

$$F - Measure = 2 \times \frac{Precision\ \times\ Sensitivity}{Precision\ + Sensitivity} \tag{11}$$

$$Accuracy = \frac{TP\ +\ TN}{TP\ +\ TN\ +\ FP\ +\ FN} \tag{12}$$

A. Lamda Architecture

In the lambda architecture, we have excluded the "Subdomain levels" feature as most of the legitimate domains have at least one subdomain and the feature had more importance over all the others after training and testing; we also used 10-fold cross-validation for each one of the classifiers.

1) Decision Tree

Table I shows the decision tree's result tested on a single model and Table II shows the 10-fold cross-validation.

TABLE. I. Decision Tree performance

Metric	Score
Sensitivity	0.9686
Accuracy	0.9556
Specificity	0.9033
F-1	0.9722

Evaluation: The decision tree appears to have the highest specificity among all the classifiers, but its accuracy and F-1 are smaller than some of those of the other ones. One way to improve the performance is to fine tune it with different parameter settings and compare the importance of each feature compare the results obtained from each method. Looking at the fit time, we see that it is the shortest compared to the other algorithms.

2) Random Forest

Table III shows the RF's result tested on a single model and Table IV shows the 10-fold cross-validation.

TABLE. III. Random Forest performance

Metric	Score
Sensitivity	0.9900
Accuracy	0.9710
Specificity	0.8942
F-1	0.9820

TABLE. IV. RF 10-fold cross-validation

Fold	Time	F-1	PRECISION	ACC.	TPR	TNR
1	2.984294	0.932720	0.978391	0.897187	0.891122	0.921405
2	2.906309	0.978306	0.975052	0.965171	0.981582	0.899497
3	3.124198	0.987321	0.980595	0.979571	0.994140	0.921273
4	3.016280	0.984469	0.974180	0.974883	0.994977	0.894472
5	3.058239	0.986307	0.977787	0.977897	0.994977	0.909548
6	3.150204	0.986722	0.977796	0.978559	0.995812	0.909548
7	2.928314	0.983797	0.976092	0.973869	0.991625	0.902848
8	2.942305	0.983817	0.974918	0.973869	0.992881	0.897822
9	2.884340	0.985056	0.976543	0.975879	0.993719	0.904523
10	3.017281	0.977064	0.973007	0.963149	0.981156	0.891122
AVG	3.001176	0.978558	0.976436	0.966004	0.981199	0.905206
STD	0.085322	0.015618	0.002144	0.023511	0.030457	0.009823

Evaluation: The random forest constitutes an improvement over the decision tree in terms of sensitivity and F-1 but not in terms of specificity. The average fit time is also way larger than that of the DT, this is one of the pitfalls of ensembles.

3) Gradient Boosting

Table V shows the Gradient Boosting's result tested on a single model and Table VI shows the 10-fold cross-validation.

TABLE. V. GBM performance

Metric	Score
Sensitivity	0.9941
Accuracy	0.9565
Specificity	0.8051
F-1	0.9734

Evaluation: Although the sensitivity, accuracy and F-1 are high, we witness the lowest specificity of all the classifiers, making the GBM not an improvement compared to the other algorithms since, as mentioned in Section III, we want to avoid false alarms (false positives) and the low specificity means more samples were misclassified.

4) Extreme Gradient Boosting

Table VII shows the XGB's result tested on a single model and Table VIII shows the 10-fold cross-validation.

TABLE. VII. XGBoost performance

Metric	Score
Sensitivity	0.9900
Accuracy	0.9710
Specificity	0.8942
F-1	0.9820

Evaluation: The overall performance of the XGBoost is great, this may be thanks to the ability to

TABLE. VI. GBM 10-fold cross-validation

Fold	Time	F-1	PRECISION	ACC.	TPR	TNR
1	2.879358	0.972884	0.954839	0.955794	0.991625	0.812709
2	2.956297	0.973506	0.955645	0.956798	0.992047	0.815745
3	3.409051	0.977641	0.958568	0.963496	0.997488	0.827471
4	2.923301	0.974862	0.952476	0.958808	0.998326	0.800670
5	3.083650	0.977814	0.960065	0.963831	0.996233	0.834171
6	2.823359	0.974601	0.953889	0.958459	0.996231	0.807370
7	2.810399	0.974780	0.955002	0.958794	0.995394	0.812395
8	2.791393	0.975540	0.958014	0.960134	0.993719	0.825796
9	2.808392	0.973760	0.953815	0.957119	0.994556	0.807370
10	2.803386	0.972039	0.954766	0.954439	0.989950	0.812395
AVG	2.928859	0.974743	0.955708	0.958767	0.994557	0.815609
STD	0.182235	0.001782	0.002279	0.002901	0.002568	0.009884

TABLE. VIII. XGB 10-fold cross-validation

Fold	Time	F-1	PRECISION	ACC.	TPR	TNR
1	0.780547	0.949087	0.974416	0.920630	0.925042	0.903010
2	0.741573	0.982128	0.975237	0.971199	0.989117	0.899497
3	0.810534	0.984870	0.975369	0.975553	0.994558	0.899497
4	0.765559	0.984304	0.971464	0.974548	0.997488	0.882747
5	0.777552	0.982609	0.972143	0.971869	0.993303	0.886097
6	0.762560	0.984660	0.974959	0.975209	0.994556	0.897822
7	0.733578	0.980522	0.970468	0.968509	0.990787	0.879397
8	0.735576	0.983375	0.976073	0.973199	0.990787	0.902848
9	0.784549	0.982988	0.974095	0.972529	0.992044	0.894472
10	0.755564	0.980124	0.969287	0.967839	0.991206	0.874372
AVG	0.764759	0.979467	0.973351	0.967108	0.985889	0.891976
STD	0.023133	0.010240	0.002217	0.015687	0.020413	0.009906

focus on the difficult to learn examples; and its average fit time is also smaller than that of the ensemble methods. Although its specificity is slightly lower that of the Decision Tree, it still has a good performance. This is therefore selected as the best model for this task as it clearly outperforms all the other models.

5) Support Vector Machine

Table IX shows the SVM result tested on a single model and Table X shows the 10-fold cross-validation.

TABLE. IX. SVM performance

Metric	Score
Sensitivity	0.9957
Accuracy	0.9586
Specificity	0.8087
F-1	0.9747

Evaluation: The SVM achieves the highest sensitivity but its specificity is too low compared to some of the others. Its accuracy is also lower than that of the Random Forest, Extreme Gradient Boosting and Multi-layer Perceptron.

6) Multilayer Perceptron

Table XI shows the MLP's result tested on a single model and Table XII shows the 10-fold cross-validation.

TABLE. XII. MLP performance

Metric	Score
Sensitivity	0.9910
Accuracy	0.9602
Specificity	0.8355
F-1	0.9755

Evaluation: First, we can see that on average, the Multi-layer perceptron is the slowest of all the classifiers, taking 40 seconds for one fit of this data set. Although its specificity and F-1 scores constitute improvements when compared to that of the SVM and

TABLE. X. SVM 10-fold cross-validation

Fold	Time	F-1	PRECISION	ACC.	TPR	TNR
1	9.374231	0.976830	0.957011	0.962157	0.997487	0.821070
2	9.129726	0.975590	0.956557	0.960147	0.995396	0.819095
3	9.318280	0.977851	0.958585	0.963831	0.997907	0.827471
4	9.180027	0.976468	0.955164	0.961487	0.998744	0.812395
5	10.033238	0.976420	0.956994	0.961487	0.996651	0.820771
6	9.558566	0.975210	0.954673	0.959464	0.996650	0.810720
7	9.768839	0.976811	0.957746	0.962144	0.996650	0.824121
8	9.774203	0.974296	0.957172	0.958124	0.992044	0.822446
9	10.349256	0.977823	0.959307	0.963819	0.997069	0.830821
10	9.251657	0.974317	0.956434	0.958124	0.992881	0.819095
AVG	9.573802	0.976162	0.956964	0.961078	0.996148	0.820800
STD	0.381051	0.001215	0.001331	0.001966	0.002033	0.005803

TABLE. XII. MLP 10-fold cross-validation

Fold	Time	F-1	Precision	ACC.	TPR	TNR
1	47.059872	0.977146	0.961118	0.962827	0.993719	0.839465
2	40.002628	0.976294	0.961820	0.961487	0.991210	0.842546
3	36.901770	0.979021	0.962394	0.965841	0.996233	0.844221
4	34.275430	0.977650	0.958199	0.963496	0.997907	0.825796
5	19.260824	0.979222	0.962783	0.966175	0.996233	0.845896
6	45.896048	0.977623	0.958921	0.963484	0.997069	0.829146
7	41.118437	0.977540	0.962272	0.963484	0.993300	0.844221
8	46.162640	0.975781	0.964797	0.960804	0.987018	0.855946
9	46.044649	0.978820	0.961616	0.965494	0.996650	0.840871
10	46.049734	0.973185	0.958943	0.956449	0.987856	0.830821
AVG	40.277203	0.977228	0.961286	0.962954	0.993719	0.839893
STD	8.177322	0.001717	0.001944	0.002741	0.003684	0.008579

GBM, they do not when compared to the XGB, RF and DT.

B. Kappa Architecture

In the kappa architecture, we have used the VFDT or Hoeffding Tree classifier for the classification task, we also tried two implementations, one with the "Subdomain levels" feature and one without it to observe how the two trees would perform.

Before using the Hoeffding Tree to make predictions for the unlabeled data set, we have streamed our labeled data set with two different levels of data imbalance: the original data set with a 10:90 ratio, and with a 20:80 ratio, then finally streamed a balanced data set with the same number of instances for both classes, i.e., 50:50.

In real data streams, the quantity of examples for each class might be changing and evolving. Therefore, the accuracy score is only useful when all classes have the same number of instances. The Kappa measurement is a more delicate measure for evaluating how streaming classifiers perform. The Kappa statistic κ was introduced by Cohen [54] and is defined by:

$$\kappa = \frac{p_0 - p_c}{1 - p_c} \tag{13}$$

The quantity p_0 is the classifier's prequential accuracy, and p_c is the probability of randomly guessing a correct prediction. If the classifier is always correct, then $\kappa = 1$. If the classifier's predictions are similar to random guessing then $\kappa = 0$. Therefore, we will observe the value of κ for every test in addition to the accuracy, precision, recall and Geometric-Mean (G-Mean), measuring how balanced the classification performances are on both the majority and minority classes, defined by (14).

$$G - Mean = \sqrt{Sensitivity \times Specificity} \tag{14}$$

Because instances are processed one at a time, we observed the mean performance of each score after a certain number of instances were processed for each level of imbalance.

In total, we have generated 6 streams:

1) *10:90 with subdomain level feature (ST1):* the data stream consisting of 10% legitimate domains and 90% malicious domains, with the "subdomain level" feature.

2) *10:90 without subdomain level feature (ST2):* the data stream consisting of 10% legitimate domains and 90% malicious domains, without the "subdomain level" feature.

3) *20:80 with subdomain level feature (ST3):* the data stream consisting of 20% legitimate domains and 80% malicious domains, with the "subdomain level" feature.

4) *20:80 without subdomain level feature (ST4):* the data stream consisting of 20% legitimate domains and 80% malicious domains, without the "subdomain level" feature, in other words, this is the same data set used in the batch learning architecture.

5) *50:50 with subdomain level feature (ST5):* the data stream consisting of 50% legitimate domains

and 50% malicious domains, with the "subdomain level" feature.

6) 50:50 with subdomain level feature (ST6): the data stream consisting of 50% legitimate domains and 50% malicious domains, without the "subdomain level" feature.

Table XIII presents the results obtained on each of these streams, all with a pretrain size of 200 samples. We observe a mean prequential accuracy higher than 90 for all the streams except from ST6 (acc = 87.54), with ST5 having the optimum accuracy (acc = 99.94). In the case of mean kappa, the optimal performance is achieved on ST5 (k = 99.89), with ST3 coming second (k = 99.77), as in the case of mean prequential accuracy (acc = 99.93). Equal optimum mean precision was obtained on ST3 and ST5 (precision = 1.00). In terms of mean recall, F-1 and G-Mean, optimum results were achieved on ST3 (recall = 99.91, f-1 = 99.95, g-mean = 99.95) with ST5 coming second with almost similar performance (recall = 99.89, f-1 = 99.94, g-mean = 99.94).

Like in the batch learning approach, it is noticed that the "subdomain levels" feature highly influence the results of models trained with streams containing the same level of data distribution, i.e., ST1-ST2, ST3-ST4 and ST5-ST6. This shows that even when instances are learned one at a time, the feature plays an important role in the correct prediction of labels.

VII. CONCLUSION AND FUTURE WORKS

In this project, we have applied two learning methods, supervised or batch learning and online learning, to classify COVID-19 related domains in our data set consisting of domains from publicly available sources with our list of malicious domains containing domains from DomainTools and PhishLabs and our list of legitimate domains with domains from search engines.

In the batch learning method, we concluded that the Extreme Gradient Boosting algorithm outperformed the other five models (Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine and Multilayer perceptron) because it achieved relatively higher scores on the data set. In the online learning method, we tested the Hoeffding Tree Classifier with streams of different levels of data distribution and feature sets and concluded that the stream ST5 was the best as the model achieved optimum mean prequential accuracy, kappa, and precision compared to the other data streams.

We then compared the Extreme Gradient Boosting's results to the ones gotten with the Hoeffding Tree on ST2, ST4 and ST6, as the "subdomain" level feature was omitted in these streams, and we noticed that the Extreme Gradient Boosting performed better than the Hoeffding Tree on the streams ST4 and ST6 but not on the stream ST2, which shows that the Hoeffding Tree kept improving with each sample it was trained from.

In the batch learning approach, future works will involve tuning the algorithms for best parameters and select the model suiting our needs to make predictions on our unlabeled data set. Those predictions will then first be compared to those made by the Hoeffding Trees pre-trained with the different levels of data distributions by using statistics such as the inter-rater reliability, kappa, to see how the classifiers agree in the predictions made for each instance. Second, those predictions will be compared with labels from VirusTotal [55], a source that we considered reliable for this purpose, to properly evaluate each model.

Finally, the best model can then be deployed as a browser addon to automatically flag and warn users that they are about to visit a malicious domains.

VIII. ACKNOWLEDGMENTS

Special thanks go to Dr. Paula Branco, Dr. Guy-Vincent Jourdan, and Dr. Herna L. Viktor for com-

TABLE. XIII. Performance of HT on ST1 to ST6

Stream	ACC.	Kappa	Precision	Recall	F-1	G-Mean
ST1	0.9775	0.8481	0.9807	0.9949	0.9878	0.8894
ST2	0.9689	0.7858	0.9741	0.9923	0.9831	0.8476
ST3	0.9993	0.9977	1.0000	0.9991	0.9995	0.9995
ST4	0.9425	0.8079	0.9458	0.9846	0.9648	0.8726
ST5	0.9994	0.9989	1.0000	0.9989	0.9994	0.9994
ST6	0.8754	0.7497	0.8448	0.9301	0.8854	0.8716

municating unique ideas, for their valuable suggestions and contributions towards the accomplishment of this course project.

REFERENCES

[1] COVID-19 coronavirus pandemic, https://www.worldometers.info/coronavirus/. Accessed Oct. 2020.

[2] Anti-Phishing Working Group (APWG), https://apwg.org/about-us/. Accessed Oct. 2020

[3] Anti-Phishing Working Group (APWG), Phishing Activity Trend Report Second Quarter, 2020 http://docs.apwg.org/reports/apwg_trends_report_q2_2020.pdf

[4] Sophie Le Page, 'Understanding the Phishing Ecosystem', 2019.

[5] Ankit Kumar Jain and B. B. Gupta, 'Phishing Detection: Analysis of Visual Similarity Based Approaches', pp. 6-10, 2017.

[6] W. Han, Y. Cao, E. Bertino, and J. Yong, "Using automated individual white-list to protect web digital identities," Expert Systems with Applications, vol. 39, no. 15, pp. 11861–11869, 2012.

[7] Y. Cao, W. Han, and Y. Le, "Anti-phishing based on automated individual white-list," in Proceedings of the 4th ACM Workshop on Digital Identity Management (DIM '08), pp. 51–60, ACM, 2008.

[8] S. Abu-Nimeh, D. Nappa, X.Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in Proceedings of the 2nd Annual eCrime Researchers Summit (eCrime '07), pp. 60–69, ACM, Pittsburgh, Pa,USA, 2007.

[9] A. ALmomani, B. B. Gupta, T.-C.Wan, A. Altaher, and S.Manickam, "Phishing dynamic evolving neural fuzzy framework for online detection 'zero-day' phishing E-mail," Indian Journal of Science and Technology, vol. 6, no. 1, pp. 3960–3964, 2013

[10] Y Zhang, J Hong and L Cranor, CANTINA: a content-based approach to detecting phishing websites, in Proceedings of the 16th International World Wide Web Conference (WWW2007), Banff, Alberta, Canada, May 8-12, pp. 639-648, 2007

[11] Scikit-learn https://scikit-learn.org/stable/

[12] A. P. E. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi, "A layout-similarity-based approach for detecting phishing pages," in *Proceedings of the 3rd International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm '07)*, pp. 454–463, September 2007.

[13] M. Dunlop, S. Groat, and D. Shelly, "GoldPhish: using images for content-based phishing analysis," in *Proceedings of the 5th International Conference on Internet Monitoring and Protection (ICIMP '10)*, pp. 123–128, Barcelona, Spain, May 2010.

[14] K. L.Chiew, E. H.Chang, S.N. Sze, andW.K. Tiong, "Utilisation of website logo for phishing detection," *Computers & Security*, vol. 54, pp. 16–26, 2015.

[15] S Sheng, B Wardman, G Warner, L Cranor, J Hong, and C Zhang. An empirical analysis of phishing black-lists, in Proceeding of the Sixth Conference on mail and Anti-Spam, CEAS, 2009

[16] Khonji, M., Iraqi, Y., & Jones, A. (2013). Phishing detection: A literature survey. IEEE Communications Surveys and Tutorials, 15(4), 2091-2121 Khonji, M., Iraqi, Y., & Jones, A. (2013). Phishing detection: A literature survey. IEEE Communications Surveys and Tutorials, 15(4), 2091-2121.

[17] Prakash, P., Kumar, M., Kompella, R. R., & Gupta, M. (2010). Phishnet: Predictive blacklisting to detect phishing attacks. In *2010 Proceedings IEEE INFOCOM* (pp. 1–5).

[18] Sharifi, M., & Siadati, S. H. (2008). A phishing site blacklist generator. In *2008 IEEE/ACS international conference on computer systems and applications* (pp. 840–843).

[19] Google Custom Search developers.google.com/custom-search

[20] Jain, A. K., & Gupta, B. B. (2018). Towards detection of phishing websites on client–side using machine learning based approach. *Telecommunication Systems, 68* (4), 687—700.

[21] Zhang, Y., Hong, J. I., & Cranor, L. F. (2007). Cantina: A content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web, WWW '07, ACM, New York, NY, USA* (pp. 639–648).

[22] Xiang, G., Hong, J., Rose, C. P., & Cranor, L. (2011). Cantina +: A feature-rich machine learning framework for detecting phishing web sites. ACM Transactions on Information and System Security, *14* (2), 1–28 20.

[23] PhishTank. *Verified phishing URL* Accessed Oct. 2020. https://www.phishtank.org/.

[24] Openfish. *Phishing dataset* Accessed Oct. 2020. https://www.openphish.com/.

[25] Jain, A. K., & Gupta, B. B. (2016). A novel approach to protect against phishing attacks at client side using auto-updated white-list. *EURASIP Journal on Information Security*, vol. 2016.

[26] Verma, R., & Das, A. (2017). What's in a url: Fast feature extraction and malicious url detection. *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics* (pp. 55-63)

[27] Sahingoz, O. K., Buber, E., Demir, O., & Diri, B. (2019). Machine learning based phishing detection from URLs. *Expert Systems with Applications*, *117*, 345-357.

[28] Yandex XML Yandex Technologies. Accessed Oct. 2020, https://yandex.com/dev/xml/doc/dg/concepts/about.html/.

[29] Extreme Gradient Boosting (XGBoost) https://xgboost.readthedocs.io/en/latest/

[30] Mitchell, T. (1997). *Machine Learning* . New York; McGraw Hill.

[31] Kittler, J., Hatef, M., Duin, R.P. W., & Matas, J.(1998). On Combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(3), 226-239.

[32] L. Breiman and A. Cutler, "Random forests-classification description," Department of Statistics Homepage,

[33] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001

[34] Vapnik, V. (1998). Statistical learning theory. New York: John Wiley

[35] Domains Tools https://www.domaintools.com/resources/blog/free-covid-19-threat-list-domain-risk-assessments-for-coronavirus-threats

[36] PhishLabs https://www.phishlabs.com/covid-19-threat-intelligence/

[37] MagicBaidu Search API https://github.com/1049451037/MagicBaidu

[38] BIng Search API https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

[39] Tranco list https://tranco-list.eu/

[40] DNSTwist https://github.com/elceef/dnstwist

[41] Freenom https://www.freenom.com/

[42] Shuang Hao, Alex Kantchelian, Brad Miller, Vern Paxson, and Nick Feamster. Predator: proactive recognition and elimination of domain abuse at time-of-registration. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communi-cations Security, pages 1568{1579. ACM, 2016.

[43] Friedman, J. H. (February 1999). "Greedy Function Approximation: A Gradient Boosting Machine"

[44] Friedman, J. H. (March 1999). "Stochastic Gradient Boosting"

[45] NearMiss – Under-sampling https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.NearMiss.html

[46] Domingo, P. & Hulten, G. Mining High-Speed data streams

[47] W. Hoeffding. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 58:13–30, 1963.

[48] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning, 2:285–318, 1997.

[49] O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, Advances in Neural Information Processing Systems 6. Morgan Kaufmann, San Mateo, CA, 1994.

[50] An introduction to Random Forest: https://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics

[51] Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961

[52] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundation. MIT Press, 1986.

[53] Scikit-multiflow Hoeffding Tree Classifier: https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html

[54] Jacob Cohen. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20(1):37–46, April 1960.

[55] Virus Total https://www.virustotal.com/gui/