Towards Detecting and Classifying Malicious URLs Using Deep Learning

Clayton Johnson[1], Bishal Khadka[1], Ram B. Basnet[1]*, and Tenzin Doleck[2]
[1]Colorado Mesa University, Grand Junction, CO 81501, USA
{cpjohnson, bkhadka}@mavs.coloradomesa.edu, rbasnet@coloradomesa.edu
[2]University of Southern California, Los Angeles, CA 90007, USA
doleck@usc.edu

Received: September 30, 2020; Accepted: December 11, 2020; Published: December 31, 2020

Abstract

Emails containing Uniform Resource Locators (URLs) pose substantial risks to organizations by potentially compromising both credentials and network security through general and spear-phishing campaigns to their employees. The detection and classification of malicious URLs is an important research problem with practical applications. With an appropriate machine learning model, an organization may protect itself by filtering incoming emails and the websites its employees are visiting based on the maliciousness of URLs contained in emails and web pages. In this work, we compare the performance of traditional machine learning algorithms, such as Random Forest, CART, and kNN against popular deep learning framework models, such as Fast.ai and Keras-TensorFlow across CPU, GPU, and TPU architectures. Using the publicly available ISCX-URL-2016 dataset, we present the models' performances across binary and multiclass classification experiments. By collecting accuracy and timing metrics, we find that Random Forest, Keras-TensorFlow, and Fast.ai models performed comparably and with the highest accuracies $> 96\%$ in both the detection and classification of malicious URLs, with Random Forest as the preferable model based on time, performance, and complexity constraints. Additionally, by ranking and using feature selection techniques, we determine that the top 5-10 features provide the best performances compared to using all the features provided in the dataset.

Keywords: Malicious URLs, Phishing URLs, Deep Learning, Web Security, Machine Learning

1 Introduction

Phishing—along with its more targeted version, spear phishing—is a social engineering attack in which the attacker attempts to compromise a user's credentials or a system by presenting itself as a legitimate business communication [38]. These communications, most commonly emails, will often contain links to websites controlled by the attacker that attempt to: 1) mimic a popular website to scrape the user's credentials, 2) install malware onto the user's system, or 3) spam the user. These links with malicious intents are called malicious URLs. According to Verizon's 2020 Data Breach Investigations Report [39], phishing attacks have been in the top three types of data breaches for the past 6 years and have been number one for the past two years, with around 96% of the social engineering attacks conducted through email. There is a clear increase in popularity with these types of attacks, highlighted by the Internet Crime Complaint Center (IC3) Public Service Announcement on Business Email Compromises (BECs) [28]. The

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 11(4):31-48, Dec. 2020
DOI:*10.22667/JOWUA.2020.12.31.031*

*Corresponding author: Department of Computer Science and Engineering, Colorado Mesa University, Grand Junction, CO 81501, USA, Tel: +1-970-248-1400

IC3's 2019 Internet Crime Report [9] reported that losses from BECs and Email Account Compromises (EACs) were in excess of $1.7 billion. The threat posed to an organization by these types of attacks is clearly substantial, with trends indicating increase in popularity and severity over time. To address the threat phishing and malicious URLs pose to businesses, many popular websites and educational institutions have added "Phishing Awareness" programs, as exemplified by the following examples: SANS Institute [16], InfoSec Institute [15], and University of Connecticut [27]. While providing training, education, and awareness on phishing attacks have become trivial and common, such efforts are not adequate to protect users. This warrants the need to apply machine learning algorithms to detect malicious URLs before they arrive at intended targets within an organization, thus decreasing the efforts required from the user. The problem of detection and classification of URLs is illustrated in Figure 1.

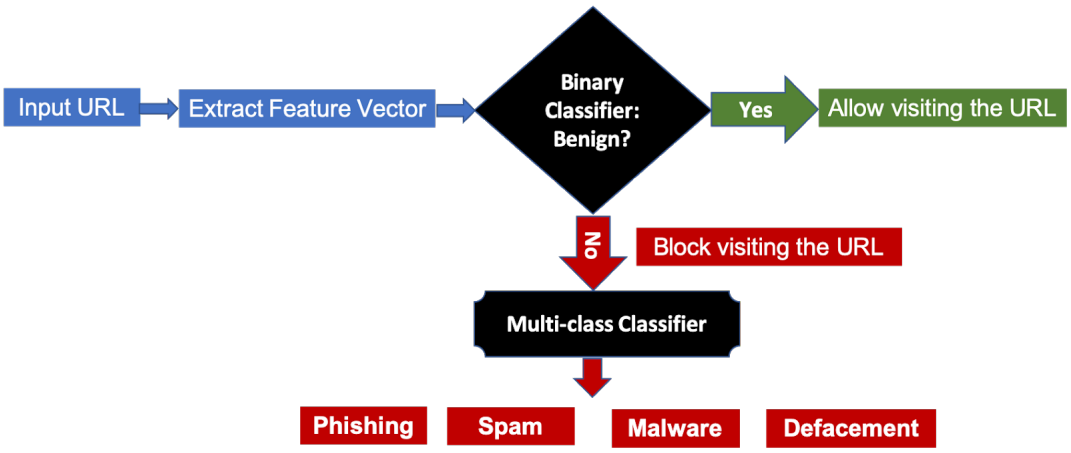

Figure 1: Detection and Classification of URLs Problem Overview

The contributions of this work are twofold: 1) we offer a direct comparison of popular deep learning frameworks (Keras and Fast.ai) against traditional machine learning algorithms (Random Forest, Classification and Regression Tree, k-Nearest Neighbor, Support Vector Machine, Logistic Regression, Linear Discriminant Analysis, AdaBoost, and Naive Bayes) in the detection and classification of malicious URLs; and 2) we expand on the literature utilizing the public ISCX-URL-2016 dataset, which contains lexical features of malicious URLs. The remainder of this research is structured as follows. The Related Works section will explore recent advancements and contributions towards the detection and classification of malicious URLs, as well as highlight any research gaps within the field. Following this, the machine learning and deep learning frameworks utilized in the work are discussed in the Frameworks section. The ISCX-URL-2016 dataset will be discussed in depth before the presentation and explanation of the experiments and results. Finally, the paper will conclude and suggest trajectories for further research.

2 Related Works

We summarize indicative studies examining malicious URLs. We follow this with a discussion of some of the patterns seen across the literature and the research gaps.

Mamun et al. [24] presented a new, public malicious URLs dataset called ISCX-URL-2016. Additionally, Mamun et al. experimented with the usage of RF, C4.5, and kNN algorithms to detect and classify malicious URLs on this new dataset by generating features directly from the URL, such as URL length, domain entropy, arguments, etc. In detecting malicious URLs, RF performed the highest with

>0.99 precision and recall; however, both C4.5 and kNN models performed with >0.97 precision and recall. When classifying malicious traffic, RF achieved the highest performance with recall and precision of 0.97. In this second classification phase, all models perform with precision and recall in the range 0.92-0.97. Finally, the authors demonstrated that the obfuscation of malicious URLs does decrease the detection and classification accuracies of the ML models used.

Cui et al. [8] proposed a system to detect malicious URLs by using NB, DT, and SVM classifiers. The reported accuracies >98.7% for all classifiers. Due to the high-performance of the models, the authors claim to have deployed the system, analyzing up to 2 TB worth of data per day. This work was extended by Zhao et al. [41], who compared the performance of RF and a Gated Recurrent Neural Network (GRU) over the same dataset used in [8] in detecting malicious URLs. Zhao et al. found that the GRU model with 98.5% accuracy outperformed the RF model with 96.4% accuracy. This trend was seen across varying sizes of training datasets (from 600 to 240,000 samples). They also presented graphs showing the distributions of each classification (Legitimate, SQL Injection, XSS Attack, Sensitive File Attack, and Directory Traversal) based on the "Number of Characters" feature. Additional statistical analysis of each feature may shed more light into understanding the malicious URLs detection problem—one of the contributions of this research study.

Choi et al. [6] experimented with the detection and classification of malicious URLs using SVM for binary classification and C4.5, Label Powerset, and kNN classifiers for multiclass classification. The features used for this system contain lexical characteristics, link structures, DNS, network traffic, and content composition information. The detection of malicious traffic demonstrated high performance with SVM's accuracy exceeding 98%. The work is claimed to be the first report of malicious URLs classification across phishing, spamming, and malware categories. Their models for multi-class classification performed with accuracies $>93\%$. Additionally, the article states that due to the details the model took into account, obfuscation techniques such as redirection, link manipulation, and fast-flux hosting would hinder the effectiveness in detecting malicious URLs.

Ma et al. [23] presented a novel technique for detecting malicious URLs by utilizing continuous, online machine learning techniques. The work experimented with Logistic Regression, SVM, NB, Confidence Weighted (CW), and Perceptron models. By combining features such as DNS information, lexical characteristics, web registration dates, etc. with blacklist information, the team was able to compile a total of 2.9 million features over the course of 100 days. The models were trained using a sliding window of two weeks' worth of training data. While all models showed decreasing error rates over the 100 day period, CW consistently maintained the lowest error rates with a minimum of 1%. Due to the dynamic landscape of the malicious URLs problem, Ma et al. addressed the need for large, fresh datasets to emphasize and encourage the continuous training of models.

Uçar et al. [37] applied two deep learning models, Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), in the detection and classification of malicious URLs. Detection of malicious URLs demonstrated high performance, with accuracies of 97.25% and 98.86% for the LSTM and CNN models, respectively. The CNN model showed an accuracy of 95.37% on the classification of malicious URLs, higher than 91.13% for LSTM. The work is significant for two reasons. First, there are few applications of deep learning in this research problem and secondly, few works have used the ISCX-URL-2016 dataset.

Vinayakumar et al. [21] evaluated the performance of multiple deep learning architectures (LSTM, CNN, Bidirectional Recurrent Structures, LSTM-CNN hybrid, and Stacked CNNs) against their proposed deep learning architecture in the detection of malicious URLs. The architectures explored were implemented through Keras, described in the Frameworks section. The two-part dataset includes data from Alexa.com, DMOZ Directory, Sophos, and other related sources. The models perform in the range of 95 - 99% accuracy over the first dataset, with LSTM as the highest performing model. The authors place an emphasis on the difference between the random-split and time-split versions of the dataset,

showing that there is a higher variance in the accuracy results from the time-split than the random-split versions of the second dataset. The models' accuracy ranges from 95-96.6% in the random-split section of the second dataset, while the time-split version has models ranging from 93-97.1% accuracy.

Sahoo et al. [33] conducted a survey of the work done within the realm of detecting and classifying malicious URLs. The work discussed in depth the multitude of issues with existing, commercial blacklist and heuristic solutions, which lead to ungeneralized, nonflexible models that are susceptible to novel or obfuscated attacks. For simplicity and security, most of the recent efforts have been focused on static analysis of URLs using various machine learning techniques. Almost half of the works analyzed utilized either lexical or host-based features [33]. There is a clear preference for these types of features within the field, while other works are spread out among HTML, JavaScript, and Context-based features. While there have been some efforts to apply deep learning architectures, such as LSTMs and CNNs to the malicious URLs problem, computational complexity is the largest constraint [33]. However, there are few works exploring and comparing the results of deep neural network frameworks against traditional machine learning algorithms such as RF, SVM, kNN, etc, in the context of the malicious URLs problem. This is the primary contribution of our work.

Hung et al. [22] performed a series of deep learning experiments and neural networks like Convolutional Neural Networks (CNN) so that the model would learn URL embedding. They use the large dataset from VirusTotal anti-virus group. The team attempted to find the limitations in identifying the malicious URLs using lexical properties like inability to identify the proper semantic meaning and patterns in URL and they use end-to-end deep learning framework to address this kind of problem. The team addressed the problem of identifying whether a URL is malicious or not by formulating a problem as a binary classification task [22]. They use the technique called character-level CNN which learns the frequency and sequence of the characters in the URL. Moreover, word-level CNN is also applied for identifying the unique words in the URL. The main goal of the paper was to use character-level and word-level CNN to address the limitations produced by previous methods and precisely identify the malicious URLs.

Birhanu et al. [10] employed a technique called BINSPECT, which is a combination of static analysis and minimalistic emulation and uses supervised learning techniques, for detecting malicious web pages. The team was able to generate accuracy of more than 97% when using this technique with low false signals [10]. Furthermore, the team also mentioned that, with this approach, it took them only 3-5 seconds to analyze a single web page.

3 Motivation

When addressing the problem, most of the works available attempt to only detect malicious URLs, while few works experiment with classifying malicious URLs. While detection is a substantial issue, the classification of these malicious URLs indicates where the direction and priority of actions within an organization should be to best protect its network. For instance, defaced URLs may require more urgent reaction due to potential credential compromises than spam URLs, which don't need immediate action.

There is a clear consensus that blacklisting malicious URLs is a poor solution since it fails to address issues such as obfuscation or novel attacks. As a consequence of this, most of the recent work has focused on applications of machine learning algorithms. Of the popular algorithms, Random Forest (RF) appears to be one of the most effective. The survey of relevant work revealed few studies that address applications of deep learning within the field and even fewer that directly compare the performance of deep learning models to traditional machine learning models. This is a key gap in the literature that the present study aims to address. Additionally, this work compares various metrics, such as training and prediction times, across multiple architectures (CPU, GPU, and TPU) to determine which model would be most practical to use in an industry setting.

The features used in the field are varied and derived from many different sources. Despite this, features may be derived from both static analysis of the URL and dynamic analysis from visiting the website. Most of the research works analyze static features for security and cost reasons. It is safer and computationally cheaper to statically analyze a suspicious URL than visit the website and execute the attacker's code, which may take an arbitrary amount of time to complete. The majority of the related works presented create custom datasets with different types of features (statistical and lexical, host-based, WHOIS, etc.), demonstrating a lack of standardization within the datasets used for the research problem. One of the reasons for our usage of the ISCX-URL-2016 dataset is to show support for usage of a standardized, lexicon-based dataset for static analysis of suspicious URLs. This work explores the effectiveness of the provided URL attributes and demonstrates the effectiveness lexical analysis may have in detecting malicious URLs, as well as the limitations.

4 Machine and Deep Learning Frameworks

4.1 fast.ai

fast.ai is one of the famous python libraries which includes various machine learning and deep learning libraries. fast.ai is a Python module that implements a high-level API to a PyTorch backend. The goal of fast.ai is to easily allow experts in other fields, such as virologists or astronomers, to implement popular deep learning techniques within their respective settings. This framework has seen multiple popular successes in research and industry [11].

4.2 Keras

Keras is a deep learning framework for Python that allows for the implementation of both TensorFlow and Theano in the backend [20]. Keras runs Tensorflow 2.0 in the backend for easy machine learning workflow and for proper data management. Keras allows for easy and fast prototyping through user friendliness, modularity, and extensibility. It usually runs seamlessly on CPU and GPU.

4.3 TensorFlow

Tensorflow was originally developed by Google and it is free to use. Tensorflow uses static computational graphs also known as the define-and-run approach. The libraries and large community resources available in the Tensorflow allows its user to build powerful and advanced machine learning models and applications. It builds and trains ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging [36]. One of the key affordances of Tensorflow is that it not only uses CPU but also GPU which helps in gaining much more computing power. Tensorflow 2.0 further uses TPU, also known as Tensor Processing Unit, which adds computational power and improves performance. Using this deep learning module, even fairly complicated models can be created with very little coding effort.

4.4 PyTorch

PyTorch is an open source machine learning framework that accelerates the path from research prototyping to production deployment [30]. This machine learning framework was first developed by Facebook. PyTorch uses dynamic computational graphs which lets the user process variable length input and output. PyTorch has many libraries like captum, pytorch geometric, and skorch which are all open source and help in the tasks like model interpretability, performing deep learning on irregular input data, and providing scikit-learn compatibility.

4.5 Scikit-Learn

Scikit-learn (sklearn)—a python library built upon Scipy, Numpy, and matplotlib—is a handy tool for predictive data analysis [29]. This cutting edge software is free to use for the public. Common machine learning tasks, such as classification, regression, and clustering, can be easily tackled using scikit-learn. For some machine learning classifiers like Support Vector Classification and Lasso, scikit-learn outperforms other python machine learning libraries [29].

The models used from the Scikit-learn module are the RandomForestClassifier, Decision Tree (optimized CART algorithm), KNeighborsClassifier, SVC (Support Vector Machine), LogisticRegression, LinearDiscriminantAnalysis, AdaBoostClassifier, and GaussianNB (Naive Bayes).

5 Dataset

This experiment presents ISCX-URL-2016 URL [24] dataset. Around 78 lexical features were extracted from URLs, and 5 URL classes such as benign, defacement, malware, phishing, and spam were labeled in the dataset. The different classes of URLs are briefly introduced below.

Benign URLs: Benign URLs are legitimate URLs that do not lead to any infectious websites and do not try to inject the user's computer with any kind of harmful malware. Benign websites may contain advertisements and adware which are typically harmless to a computer.

Defacement URLs: Website defacement usually means changing a certain aspect of the website such as it's visual appearances and some contents on it. Hacktivists try to deface a website for numerous reasons [32]. This kind of act is done when some content in the web page needs to be changed without the permission of the original owner of the website which technically means penetrating a website.

Malware URLs: Malware URLs take a user to the malicious website that typically installs some malware on that user's device which can be used for identity theft, corrupting files, and even logging keystrokes. Malware can indirectly be a dangerous software that can harm a computer and steal someone's private information [25]. Some threats such as harmful biological agents, a terrorist cell intent on disrupting operations, etc. can be considered as malware [26]. Some of the examples of malware are ransomware, spyware, scareware, among others.

Phishing URLs: Phishing URLs conventionally entice a user to visit a fake website and will try to steal as much information they can get from the user. Sometimes a user can easily be led to phishing websites just by having a typo in a URL. Phishing can be defined as the intent of the hacker to steal some private information like credit card number and other digital identity by employing social engineering techniques [40].

Spam URLs: Spam is a way of sending unsolicited emails to the user with the intent of advertisements or for serious harm to the computer [19]. Spam URLs are usually seen in the spam email. Some spam URLs can be harmful and can infect the computer of the user with spyware and adware.

5.1 Lexical Analysis

Malicious URLs may contain some pattern in their URL text which gives us a hint that the URL is not legitimate. Various lexical features such as query length, domain token count, path token count, URL length, domain length, path length, and many more were used in the experiment for better performance and results. Let's consider the following URL to extract lexical features, e.g.:

http://www.example.site.com/path_dir1/path_dir2/file.php

The URL is 56 characters long, thus the 'URLLen' feature is 56. The length of the domain name "domainlength" example.site.com is 16, "domainUrlRatio" is 0.2857, and "NumberofDotsinURL" is 4. This process of extracting all 78 lexical features is thoroughly discussed in [24].

As discussed previously in the Related Works section, traditional blacklisting techniques have clear disadvantages in the detection and classification problems. Thus, taking a closer look at lexical analysis would be a better choice for identifying those URLs.

5.2 Data Preprocessing

Original dataset had a total sample of 36,707. From this, many entries were NaN, infinity, or empty. All the entries and attributes with NaN, Infinity, and missing values were eliminated from the dataset with the aim of getting more precise results. During the data cleansing process, around 17K rows with NaN and redundant values were dropped from the dataset. In addition, 7 attributes of columns with NaN values were eliminated leaving only 72 attributes in the dataset. Table 1 illustrates the number of samples before and after data cleanup.

URL Type	Raw Samples	Filtered Samples
Benign	7,781	2,709
Defacement	7,930	2,477
Malware	6,712	4,440
Phishing	7,586	4,014
Spam	6,698	5,342

Table 1: Number of samples before and after data cleanup

6 Experiment and Results

The experiments demonstrate a two-layered approach to the malicious URLs research problem: (1) Experiment 1 involves the detection of malicious URLs and (2) Experiment 2 involves the classification of malicious URLs. Experiment 1 is a binary classification problem with two classes: Benign and Malicious. All non-benign categories of URLs are labelled as malicious. Experiment 2 is a multi-class classification problem where malicious URLs are categorized into specific classes such as: Defacement, Malware, Phishing, and Spam.

Both the experiments utilize the same set of machine learning models: Random Forest [3] (RF), Decision Tree/CART [4] (DT), k-Nearest Neighbors [7] (kNN), Support Vector Machine (SVM), Logistic Regression (LR), Linear Discriminant Analysis (LDA), AdaBoost [13] (AB), Naive Bayes (NB). In addition, two deep learning models, Fast.ai [14] and Keras-TensorFlow [20] [36] [1], are also used. The models are trained and evaluated on the ISCX-URL-2016 dataset using stratified 10-fold cross validation. All experiments were conducted on the free tier of Google Colaboratory system using Jupyter Notebooks. The Jupyter Notebooks and scripts used for the experiments can be found on GitHub.com [2].

6.1 Performance Metrics

Various metrics are collected to compare and determine the most appropriate model meeting real-world deployment constraints. Initially, all models are trained and evaluated on both Experiments 1 and 2, generating accuracy using 10-fold cross validation. Confusion matrices are presented to clearly demonstrate the performance of the DNN models. An additional experiment is performed on the highest-performing traditional machine learning and the DNN algorithms in which the benign samples are removed from the multi-class classification problem. The goal of the experiment is to see how the best classifiers would

perform on multi-class classification of just the malicious URLs. Then, time metrics are calculated on these three models. These time metrics describe two different characteristics of the models. These time values are collected across CPU, GPU, and TPU architectures to further assist an organization's investment decisions in projects in which architecture is a consideration.

The first time metrics used to evaluate the models is the total time required to train each model. This training time metric is calculated by summing the total training time elapsed for each fold. Training time excludes object instantiation and testing since the focus is on training specifically. Even though training is often described as a single event, training time is included as a metric because retraining with new data may occur multiple times for a model implemented by a system. Thus, the total amount of time required to train a model has a direct impact on the organization.

The second time metric is the average time required for a model to predict the classification of a provided sample. If a model was deployed and used to predict the malevolence of a URL, the time required to generate this prediction has a clear impact on the ability of the organization to filter URLs. This effect is exponential for larger organizations with growing attack surfaces. For simplicity, we assume that each deployed model would predict on a single sample at a time, as opposed to batches of samples. Additionally, since this metric is calculated by timing the 'predict' method of each object on samples from the ISCX-URL-2016 dataset, we do not report the time required to generate the features used in the dataset. However, time required to generate features may be minimized by balancing the trade-off between the number of features used and reported accuracy, as discussed in the feature analysis section.

6.2 Results

As seen in Table 2, Random Forest demonstrates superior accuracy among all machine learning algorithms used for both the experiments. This result is well documented in the literature [21, 40] and further supported by our experiments. Both DNN models show high accuracy in Experiment 1, performing similar to DT, kNN, and AB, while still being eclipsed by RF. Experiment 2 sees performance decreases for all models except Fast.ai, with both Fast.ai and RF performing comparably. Keras-TensorFlow's accuracy decreased substantially to 91.5%, around five percentage points lower than that from RF and Fast.ai. An additional observation is the higher standard deviation for Keras-TensorFlow.

Classifier	Binary-Class Accuracy (%)	Multi-Class Accuracy (%)
RF	$\mathbf{98.68 \pm 0.22}$	$\mathbf{96.26 \pm 0.53}$
DT	97.63 ± 0.24	92.81 ± 0.62
kNN	97.47 ± 0.39	92.52 ± 0.79
SVM	93.96 ± 0.60	80.77 ± 0.63
LR	90.50 ± 0.30	69.54 ± 0.97
LDA	94.34 ± 0.33	82.31 ± 0.78
AB	96.21 ± 0.39	76.58 ± 1.60
NB	68.73 ± 0.83	59.55 ± 0.91
fast.ai	$\mathbf{96.88 \pm 0.33}$	$\mathbf{96.77 \pm 0.51}$
Keras-TensorFlow	$\mathbf{97.13 \pm 1.93}$	$\mathbf{91.45 \pm 2.94}$

Table 2: Accuracy Results for Binary and Multi-Class Classification

Removing the benign samples contained in the multi-class dataset, as seen in Table 3, appears to have little effect on the performance of the classifiers. While all three classifiers appear to perform better without the benign samples, the performance is within one standard deviation of the previous results. Thus, this improvement appears to be insignificant. Nevertheless, RF, Fast-AI, and Keras-TensorFlow classifiers clearly demonstrate high accuracy metrics when classifying malicious URLs.

Classifier	Accuracy (%)
RF	96.99 ± 0.50
fast.ai	97.55 ± 0.37
Keras-TensorFlow	93.81 ± 2.34

Table 3: Top Classifier's Accuracies Results for Multi-class Malicious URL Classification (without Benign Samples)

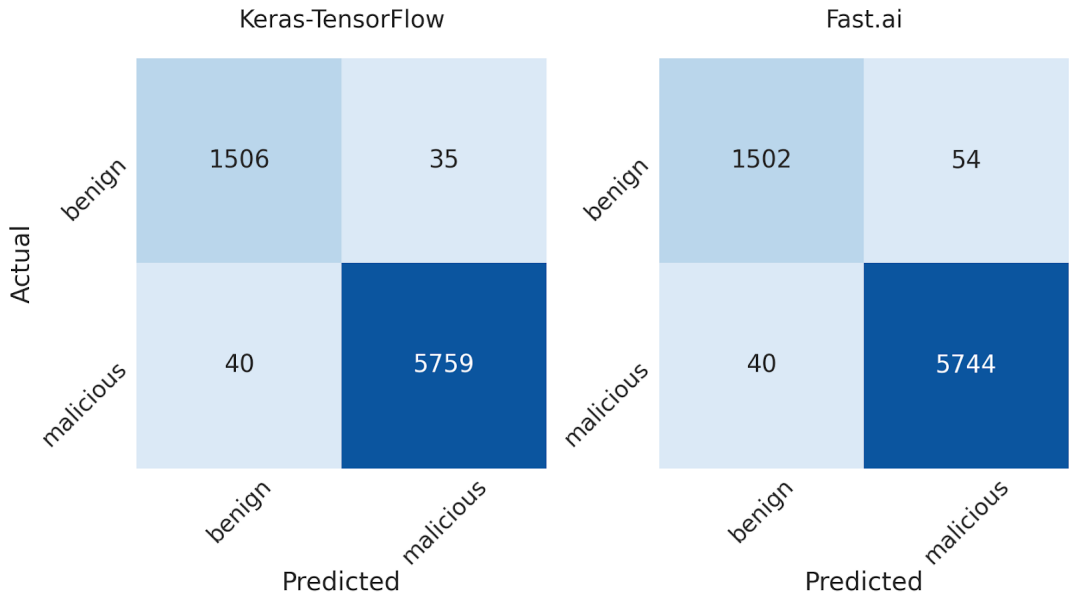

Figure 2: Confusion Matrices for Keras-TensorFlow and Fast.ai DNN Models for Experiment 1

The confusion matrices for the deep learning models are presented in Figure 2 and Figure 3. While an overwhelming majority of the samples are classified correctly—seen through the main diagonal of the matrix— the Phishing URL classification has the clearest rate of failure. Both deep learning models demonstrate a pattern of both over- and under-predicting Phishing URLs.

Classifier	Experiment	CPU	GPU	TPU
RF	Binary	75.83	**61.58**	95.19
	Multi	87.26	71.02	106.83
fast.ai	Binary	323.36	221.52	360.47
	Multi	320.91	**220.00**	362.29
Keras-TensorFlow	Binary	445.78	135.63	**123.17**
	Multi	451.37	135.65	124.09

Table 4: Training Time across Runtime Environments in Seconds

Table 4 presents the total time taken for Random Forest and the DNN models to train across the 10 folds on a CPU, GPU, and TPU run-time environments. Keras-TensorFlow demonstrates its lowest train times on the TPU, as expected since Google develops both TensorFlow and the TPU architecture [18]. However, the best metric for Keras-TensorFlow is still twice the fastest time recorded for RF. Both RF and Fast.ai have their best training times on the GPU, which is likely because there is currently no support or documentation for the TPU architecture through Scikit-Learn or Fast.ai's PyTorch version (Fast.ai

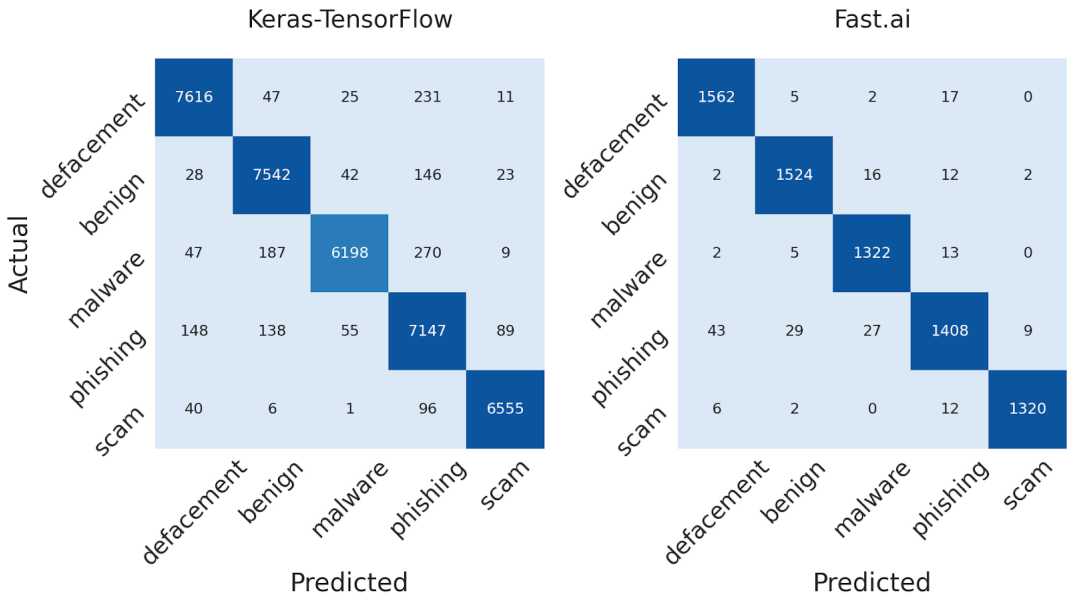

Figure 3: Confusion Matrix for Keras-Tensorflow (left) and Fast.ai (right) for Multi-Class Classification Experiments

Classifier	Experiment	CPU	GPU	TPU
RF	Binary	8.12 ± 0.77	$\mathbf{6.61 \pm 0.63}$	7.99 ± 0.74
	Multi	8.30 ± 1.09	6.66 ± 0.73	8.02 ± 0.66
fast.ai	Binary	53.17 ± 9.95	$\mathbf{44.29 \pm 6.65}$	54.59 ± 8.888
	Multi	54.62 ± 17.31	44.34 ± 4.98	54.53 ± 11.05
Keras-TensorFlow	Binary	38.53 ± 4.71	$\mathbf{28.81 \pm 5.51}$	41.52 ± 4.30
	Multi	40.20 ± 5.56	28.54 ± 4.48	41.49 ± 4.32

Table 5: Average Time to Predict Classification of a Sample in Milliseconds

currently implements PyTorch v1, however Fast.ai v2 claims it will support TPU through PyTorch v1.3+) [31] [12] [14]. While Keras-TensorFlow trains faster than Fast.ai, there appears to be a large performance hit, especially for Experiment 2. There is little change in performance times based on the experiment conducted.

The times presented in Table 5 show the average time for the RF and DNN models to predict a classification for a given sample. As expected, there is a uniform drop in average time from CPU to GPU. The TPU architecture, on the other hand, appears to have mixed results on the prediction times, leading to all three models showing the best prediction times on the GPU. Finally, Random Forest appears to be the most consistent with standard deviations less than 1.09 ms. This result is in extreme contrast to the DNN models, which experience standard deviations ranging from 4.30 ms to 17.31 ms. These metrics appear to have mixed results depending on the experiment conducted.

6.3 Feature Analysis

The features in the ISCX-URL-2016 dataset are analyzed to shine further light onto the subject of developing efficient mechanisms to detect and classify malicious URLs. There are a total of 78 features available from the dataset. These features naturally vary in their predictive capabilities and thus their

usefulness for a machine learning algorithm. By applying the chi-squared test [29] [35] [34], we rank the features with highest correlation to the classification of a malicious URL sample. The chi-squared feature ranking technique is chosen for its simplicity and availability through sklearn [35]. Additionally, the p-value calculated by the test for each feature clearly indicates its importance in classification problems, where high p-values indicate independence from the target classification and are thus poor-quality features and vice-versa. The effect of this process is two-fold. Using this feature selection technique, we decrease noise within a dataset and decrease the resources required to generate a prediction on a sample in a real-world setting. If a model handles noisy data poorly and thus performs poorly, this will help remedy its sensitivity to the data. Additionally, when trying to determine if a URL in an email or a website is malicious, less time and computation is spent generating characteristics that are unused or not as useful.

Rank	Binary-Class Feature	Binary-Class p-value	Multi-Class Feature	Multi-Class p-value
1	fileNameLen	1.406341e-70	Entropy_Afterpath	0
2	domain_token_count	1.636373e-41	argPathRatio	1.397820e-290
3	tld	1.636373e-41	NumberRate_AfterPath	1.983143e-290
4	SymbolCount_Domain	1.678641e-41	NumberRate_Domain	5.645027e-279
5	Entropy_Afterpath	3.584071e-41	ArgUrlRatio	3.924478e-272
6	delimeter_path	9.681864e-40	Extension_DigitCount	6.874504e-171
7	argPathRatio	9.901046e-38	dld_getArg	1.534782e-147
8	Entropy_Filename	2.679810e-37	ldl_getArg	1.714726e-147
9	Entropy_DirectoryName	7.487584e-36	Query_DigitCount	1.118545e-135
10	Filename_LetterCount	3.891527e-33	LongestVariableValue	2.788998e-135
...				
69	Entropy_URL	0.161457	ldl_domain	7.808014e-11
70	isPortEighty	0.248489	domainlength	8.353102e-10
71	Directory_LetterCount	0.265838	Entropy_Domain	6.009946e-09
72	pathDomainRatio	0.305007	host_letter_count	1.007751e-08
73	dld_domain	0.339682	avgpathtokenlen	1.192005e-07
74	NumberRate_URL	0.442371	isPortEighty	2.881362e-05
75	sub-Directory_ LongestWordLength	0.587184	sub-Directory_ LongestWordLength	3.440676e-04
76	Path_LongestWordLength	0.868772	dld_domain	4.124519e-04
77	charcompvowels	0.911420	Path_LongestWordLength	1.380409e-02
78	avgdomaintokenlen	0.993084	Entropy_URL	1.272731e-01

Table 6: Feature Rankings across Binary and Multi-Classification Experiments using Chi-Squared Test

Table 6 presents the results of applying the Chi-Squared test on the feature sets for the binary and multi-class datasets. It is interesting to note that out of the top ten highest rated features for both datasets, only two are present in both (20%). Conversely, 50% of the features in the lowest-rated—the bottom 10 features—are shared. While the best features chosen for a model are highly dependent on the problem (detection vs classification), it appears that there is a consensus on which features to not use. The "Entropy_AfterPath" feature's p-value is rounded off since the smallest float available for the distribution of Python used (3.8.3) is 1.0e-308. The 'ISIpAddressInDomainName' feature was removed from both datasets since there was no variation in the data.

Three of the top ten features for the binary classification problem describe the entropy of various segments of a URL. The high value of these features is also reported in [17], which reported a 20% increase in malicious URL detection combined with a decrease in false negative rates after deploying the model.

Figures 4 and 5 present the distribution graphs of the top four rated features from Table 6 for the

multi-class and binary-class datasets, respectively. One interesting note for the 'argPathRatio' graph in Figure 4 is that the Defacement and Spam classifications tend towards larger values and larger variation, while Phishing links tend towards much smaller values with the smallest variation out of all the classifications. This trend may be due to the fact that many malicious URLs may attempt to obfuscate their intent through long arguments, while seemingly benign URLs may have shorter arguments with longer paths. The other features shown present seemingly more complex relationships across the data. This is expected given the increased number of classifications.

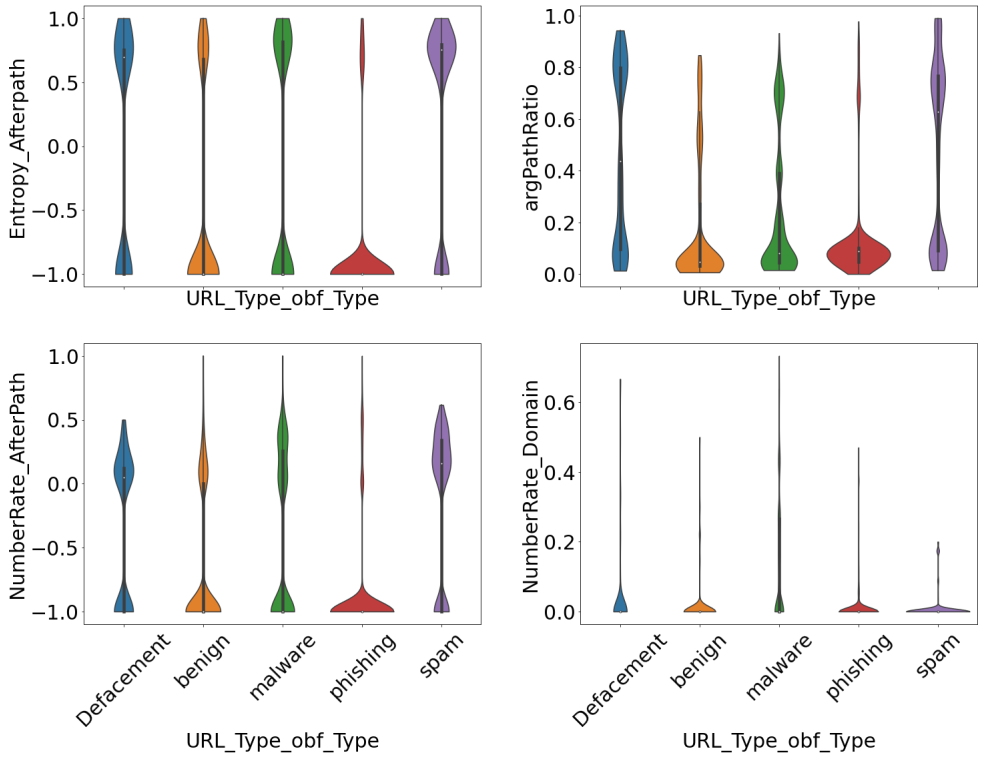

Figure 4: Distribution Plots of Top Four Multi-class Features

The binary-class features presented in Figure 5 show intriguing trends. Starting with 'fileNameLen', one may expect to observe that malicious URLs have longer filenames for obfuscation reasons; however, the distribution plot indicates otherwise. Further, there is wider variation in the benign URL filename lengths than malicious URLs. There are clear trends in 'domain_token_count', 'tld', and 'SymbolCount_Domain' features, all showing malicious URLs tend to have more items than benign URLs. This result is supported by [24] [17] [5], which report higher presence of special characters and usage of multiple top-level domains (tld) in malicious URLs.

Figures 6 and 7 present the accuracies of RF and DNN models with varying numbers of features, using the same stratified 10-fold cross validation methodology as before. The features included are in order of rank as presented in Table 6; as such, the models are trained on the rank 1 feature, then the rank 1 and rank 2 features, etc until all 78 features are included (note that the 'ISIpAddressInDomainName' feature is omitted, for no variance occurs). It is expected that the models perform with higher accuracy as we increase the number of features available; however, the goal is to determine the optimal number of features such that the accuracy of the model is not penalized substantially. Figure 6 presents data from one feature up to 20 features for both binary and multi-class problems to increase the data's resolution, while Figure 7 illustrates the model accuracies as we increase the feature count up to 78. We briefly

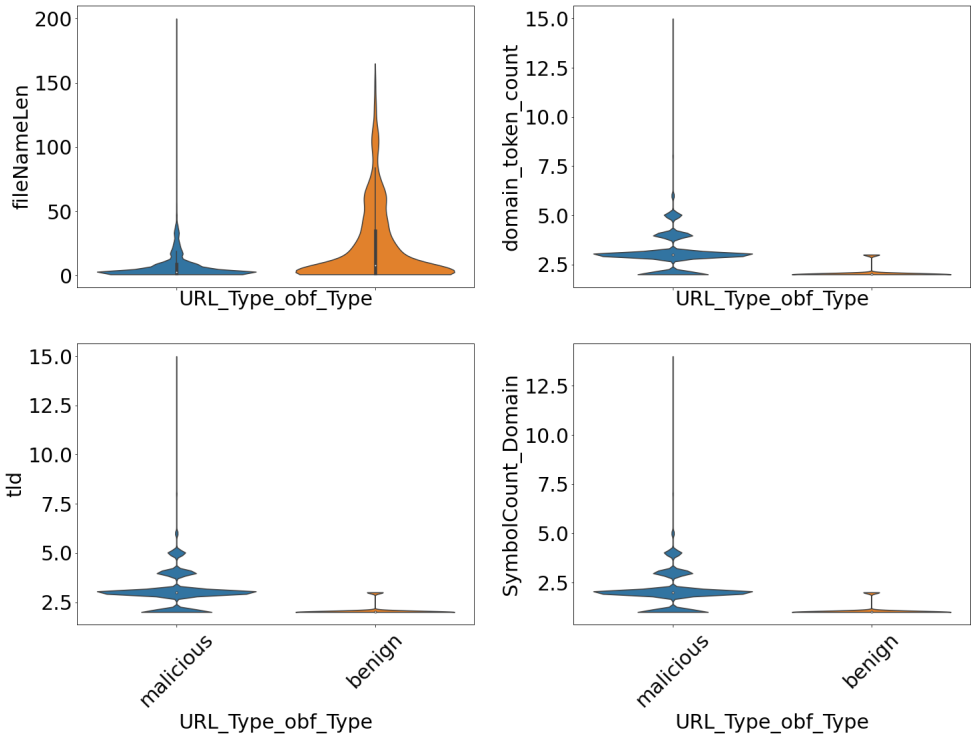

Figure 5: Distribution Plots of Top Four Binary-class Features

analyze the results in each figure below.

The multi-class problem analyzed in Figure 6 shows the expected upward trend for all models across the first 20 features; however, most of this upward growth occurs in the first five features. Random Forest demonstrates the significance of these first five features and slowly increases as we add more features. For the RF model, this is a clear indication of diminishing returns as the feature count increases. Both Keras and Fast.ai severely underperform compared to RF for the first 20 features. The Keras-TensorFlow model has the largest growth in accuracy from 1-5 and 15-20 features. Fast.ai increases performance by 30 percentage points within the first 10 features, however this accuracy stays constant with the addition of another 10 features. Finally, the variation of the accuracies across the 10 folds is shown, with the Keras-TensorFlow model demonstrating extremely high variation compared to both Fast.ai and RF.

In contrast to Figure 6, the accuracies of the models on the binary classification problem indicate that the models perform similarly and require fewer features. While RF still demonstrates superior accuracy, Keras is consistently within 5 percentage points. Surprisingly, these models were able to perform with 85-90% accuracy with only the first feature. Fast.ai, conversely, shows very poor performance with the first feature, but the performance slowly increases as the feature size increases. Additionally, all models show smaller variation across the folds compared to the data seen in the multiclass problem (Figure 6).

Figure 7 show the change in accuracies for each model from one feature up to all 78, with half the data resolution of Figure 6. We see that the accuracies of all the models in Figure 7 increase with the addition of more features, as expected. However, while Fast.ai shows substantial improvement in performance from 20 to 50 features, neither Keras-TensorFlow nor RF dramatically increase in performance. The issue of high variation across the folds continues for Keras-TensorFlow, while only being an issue for Fast.ai between 10 and 25 features.

While it is expected that the models will perform with higher accuracy when we increase the features available to the algorithms, Figure 7 presents slightly different trends. While RF, Keras-TensorFlow,

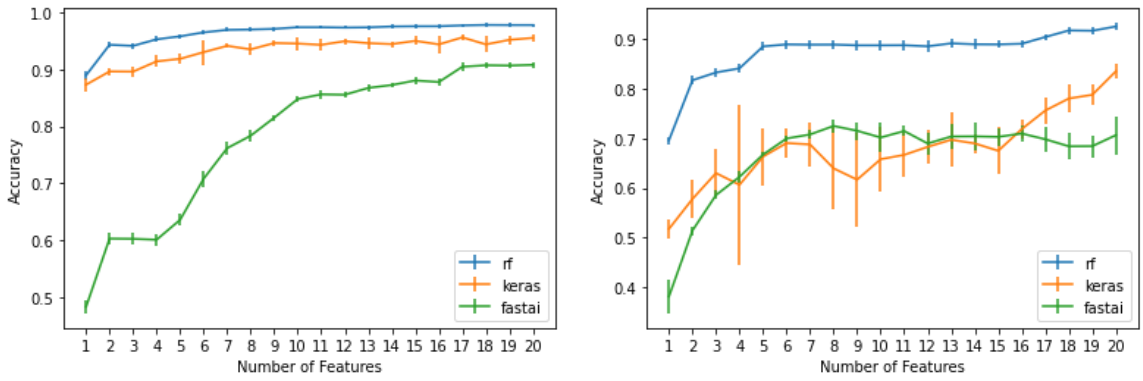

Figure 6: Binary Classification (left) and Multi Classification (right) for Increasing Number of Features up to n=20

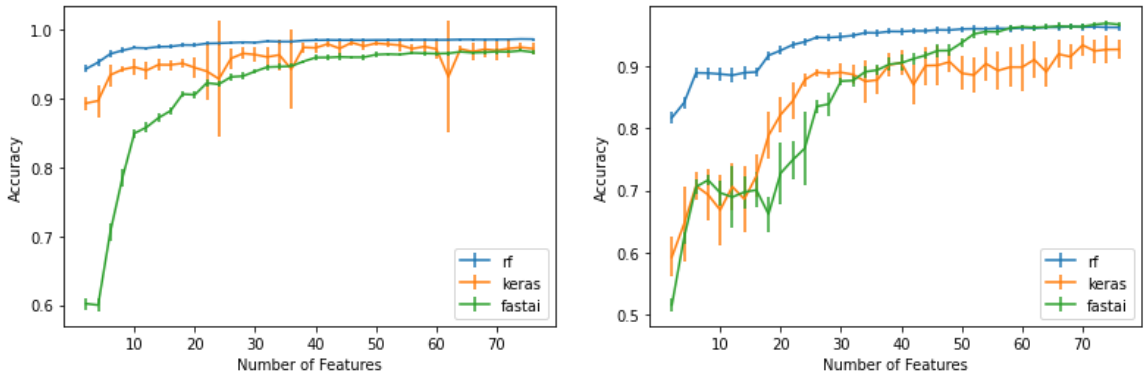

Figure 7: Binary Classification (left) and Multi Classification (right) for Increasing Number of Features up to n=78

and Fast.ai all show increases in performance from 1 to 40 features, the addition of more features appears to have a detrimental effect on RF and Keras-TensorFlow models. While the Keras-TensorFlow model shows high variance within the multiclass problem, the binary classification problem sees this for the addition of specific features. In order of appearance from left to right, the features that appear to have substantial, negative effects on the performance of the Keras-TensorFlow model are 'domainUrlRatio', 'ldl_getArg', 'File_name_DigitCount', 'File_name_DigitCount', 'LongestVariableValue', and 'subDirLen'.

7 Discussion

The current research sought to explore the effectiveness of the provided URL attributes and demonstrate the effectiveness lexical analysis may have in detecting and classifying malicious URLs, placing an emphasis on practicality for an industrial setting. This experimental research mainly focused on the detection (Experiment 1) and classification (Experiment 2) of various types of URLs based on lexical analysis through binary and multiclass classification experiments, with an emphasis on the comparison of popular deep learning models with traditional machine learning algorithms. Experiment 1 results demonstrated higher performance accuracy overall, with an increase of 8-10% on average across all models. While Experiment 2 results showed lower performance with the average accuracy around >

85%. Random Forest, Keras-TensorFlow, and Fast.ai consistently performed the best of all the models experimented, with $> 96\%$ accuracy in both the experiments.

By collecting the training and prediction times, in conjunction with the feature analysis, we conclude that, despite their popularity, deep neural networks are inferior to Random Forest due to their higher variance, count of features required to match RF performance, complexity, and overall amount of time to train and predict when deployed. To minimize the effort required to potentially deploy a RF model, the feature set can be reduced down to 5-10 features with minimal cost to performance. While both Keras-TensorFlow and Fast.ai are popular DNN frameworks, their deployment over RF requires more resources which may be better spent elsewhere within an organization.

Overall, it is clear from the results that Random Forest is the most appropriate model for deployment in an organization's detection system. RF was a model that demonstrated lower complexity, as seen in its lower training (Table 4) and prediction times (Table 5), and higher performance as seen in Tables 2 and 3. Further, RF typically performed up to or over 90% accuracy within the top five features for both binary and multi-class problems, whereas both DNN models required up to 30-40 features to match RF's performance with only five in the multi-class problem (Figures 6 and 7).

The results acquired from the deep neural network models indicate further work is required to clearly demonstrate one's superiority over another. While the accuracy of the Fast-AI model exceeded that of Keras-TensorFlow (Tables 2 and 3), Fast-AI experienced a substantial performance hit for both training and sample prediction times (Tables 4 and 5, respectively). With the current work, a preference of one DNN model over the other would indicate the priorities required from the model: accuracy at the cost of time dictates Fast-AI superior while low latency at the cost of accuracy prefers the Keras-TensorFlow model.

Additionally, as the final contribution of this work, the feature analysis of the lexical-based ISCX-URL-2016 dataset shows the importance of specific characteristics of these malicious URLs. The main conclusion from this section of the work clearly indicates a higher need for more features in the multi-classification problem than the binary classification problem. This is clearly seen from the p-values presented in Table 6 and the slower increase in accuracies in Figures 6 and 7. While all three models showed large improvements within the first 5-10 features, as ranked by their p-values, there was a drastic improvement in the accuracies when features ranked 15-25 were included.

8 Conclusion and Future Work

This work explored the applications of machine learning and deep learning models in detecting and classifying malicious URLs. By collecting performance accuracy, confusion matrices, and both training and predicting times for the Random Forest, Keras-TensorFlow, and Fast-AI models, this study concludes that Random Forest is the best model to deploy by organizations seeking to build an URL filter application, or those wishing to incorporate machine learning techniques to improve existing ones. Additionally, this study reports the specific lexical features contained within URLs may be used to minimize overhead cost of a deployed model.

Some limitations attached to the present study could motivate further research; e.g., our team did not exhaustively explore all the network configurations and hyperparameters available for DNNs which may potentially improve their performances. While these improvements may lead to succeeding RF's reported accuracy, there is an impact on training and testing times as well as additional drawback of overfitting models thus reducing their real-word generalizability. Finally, we did not deploy and investigate the efficacy of the models with further experiments as explored in [17]; we leave this as our future research work. Importantly, we feel that more research on this front is needed to better bridge the gap between academic research and industrial applications with the goal of reducing detrimental economic impacts of

malicious URLs on organizations in various industries.

Acknowledgments

This research project was supported by the state of Colorado through funds appropriated for cybersecurity law dubbed "Cyber Coding Cryptology for State Records." Any opinions, findings and conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding sources. The authors would like to thank Robert Dunsky, Nathan Bellew and Karen Angels for helping with some of the early versions of the experiments.

References

[1] M. Abadi, A. Agarwal, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv*, abs/1603.04467, March 2016.

[2] R. Basnet. Deep learning malicious urls. "https://github.com/rambasnet/DeepLearningMaliciousURLs", [Online; accessed on September 10, 2020], 2019.

[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, October 2001.

[4] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[5] Z. Chen and J. Szurdi. Cybersquatting: Attackers mimicking domains of major brands including facebook, apple, amazon and netflix to scam consumers. https://unit42.paloaltonetworks.com/cybersquatting/ [Online; accessed on September 15, 2020], 2020.

[6] H. Choi, B. Zhu, and H. Lee. Detecting malicious web links and identifying their attack types. In *Proc. of the 2nd USENIX Conference on Web Application Development (WebApps'11), Portland, Oregon, USA*, page 11. USENIX, June 2011.

[7] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, January 1967.

[8] B. Cui, S. He, X. Yao, and P. Shi. Malicious url detection with feature extraction based on machine learning. *International Journal of High Performance Computing and Networking*, 12(2):75–84, August 2018.

[9] Cybersecurity and I. S. Agency. Fbi releases ic3 2019 internet crime report. "https://us-cert.cisa.gov/ncas/current-activity/2020/02/12/fbi-releases-ic3-2019-internet-crime-report" [Online; accessed on May 08, 2020], 2019.

[10] B. Eshete, A. Villafiorita, and K. Weldemariam. Binspect: Holistic analysis and detection of malicious web pages. In *Proc. of the 8th International ICST Conference (SecureComm'12), Padua, Italy*, volume 106 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 149–166. Springer, Berlin, Heidelberg, September 2012.

[11] fast.ai. About. "https://www.fast.ai/about" [Online; accessed on July 29, 2020].

[12] fast.ai. fastai v1 for pytorch: Fast and accurate neural nets using modern best practices. https://www.fast.ai/2018/10/02/fastai-ai/ [Online; accessed on September 10, 2020].

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, August 1997.

[14] J. Howard and S. Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, February 2020.

[15] Infosecinstitute. Infosec iq, power to your people. "https://www.infosecinstitute.com/iq/", [Online; accessed on August 18, 2020].

[16] S. Institute. Robust phishing awareness simulation training that changes behavior. "https://www.sans.org/security-awareness-training/products/phishing", [Online; accessed on August 18, 2020].

[17] A. Joshi, L. Lloyd, P. Westin, and S. Seethapathy. Using lexical features for malicious url detection–a machine learning approach. arXiv:1910.06277, October 2019. https://arxiv.org/abs/1910.06277][Online; accessed on September 15, 2020].

[18] N. P. Jouppi, C. Young, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. of the 44th Annual International Symposium on Computer Architecture (ISCA'17), Toronto, Canada*, pages 1–12. ACM, June 2017.

[19] A. Karim, S. Azam, B. Shanmugam, K. Kannoorpatti, and M. Alazab. A comprehensive survey for intelligent spam email detection. *IEEE Access*, 7:168261–168295, November 2019.

[20] Keras.io. Keras: The python deep learning api. `https://keras.io` [Online; accessed on August 09, 2020].

[21] S. KP, M. Alazab, et al. Malicious url detection using deep learning. `https://www.techrxiv.org/articles/preprint/Malicious_URL_Detection_using_Deep_Learning/11492622` [Online; accessed on September 15, 2020], January 2020.

[22] H. Le, Q. Pham, D. Sahoo, and S. Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. arXiv:1802.03162, March 2018. `https://arxiv.org/abs/1802.03162` [Online; accessed on September 15, 2020]].

[23] J. Ma, L. Saul, S. Savage, and G. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09), Montreal Quebec, Canada*, pages 681–688. ACM, June 2009.

[24] M. Mamun, M. Rathore, A. Lashkari, N. Stakhanova, and A. Ghorbani. Detecting malicious urls using lexical analysis. In *Proc. of the 10th International Conference on Network and System Security (NSS'16), Taipei, Taiwan*, volume 9955 of *Lecture Notes in Computer Science*, pages 467–482. Springer, September 2016.

[25] Microsoft. Defining malware: Faq. `"https://docs.microsoft.com/en-us/previous-versions/tn-archive/dd632948(v=technet.10)"` [Online; accessed on August 13, 2020].

[26] T. Nash. An undirected attack against critical infrastructure: A case study for improving your control system security. `"https://us-cert.cisa.gov/sites/default/files/recommended_practices/CaseStudy-002.pdf"` [Online; accessed on August 13, 2020], 2005.

[27] U. of Connecticut. Protect yourself from future phishing scams. `"https://phishingeducation.uconn.edu/"`, [Online; accessed on August 18, 2020], 2020.

[28] F. B. of Investigation. Business email compromise the $26 billion scam. `"https://www.ic3.gov/media/2019/190910.aspx"`, [Online; accessed on August 18, 2020], 2019.

[29] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[30] PyTorch. Pytorch. `https://pytorch.org` [Online; accessed on August 09, 2020].

[31] PyTorch. Pytorch 1.3 adds mobile privacy quantization and named tensors. `"https://pytorch.org/blog/pytorch-1-dot-3-adds-mobile-privacy-quantization-and-named-tensors/#speech-extensions-to-fairseq"`, [Online; accessed on October 09, 2020].

[32] M. Romagna and N. van den Hout. Hacktivism and website defacement: Motivation, capabilities and potential threats. In *Proc. of the 27th Virus Bulletin International Conference, Madrid, Spain*, October 2017.

[33] D. Sahoo, C. Liu, and S. Hoi. Malicious url detection using machine learning: A survey. arXiv:1701.07179, August 2019. `https://arxiv.org/abs/1701.07179` [Online; accessed on August 13, 2020].

[34] Scikit-Learn. 1.13. feature selection - univariate feature selection. `"https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection"` [Online; accessed on September 15, 2020], 2007.

[35] Scikit-Learn. sklearn.feature_selection.chi2. `"https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2"` [Online; accessed on September 15, 2020], 2007.

[36] TensorFlow. Tensorflow. `https://www.tensorflow.org` [Online; accessed on August 09, 2020].

[37] E. Uçar, M. Incestaş, and M. Ucar. A deep learning approach for detection of malicious urls. In *Proc. of the 6th International Management Information Systems Conference (IMISC'19), Istanbul, Turkey*, pages 12–20, October 2019.

[38] A. Van der Merwe, M. Loock, and M. Dabrowski. Characteristics and responsibilities involved in a phishing attack. In *Proc. of the 4th International Symposium on Information and Communication Technologies (WISICT'05), Cape Town, South Africa*, pages 249–254. Trinity College Dublin, January 2005.

[39] Verizon. 2020 data breach investigations report. `https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf` [Online; accessed on May 08, 2020].

[40] R. Verma and A. Das. What's in a url: Fast feature extraction and malicious url detection. In *Proc. of the 3rd ACM on International Workshop on Security and Privacy Analytics (IWSPA'17), Scottsdale, Arizona, USA*, pages 55–63. ACM, March 2017.

[41] J. Zhao, N. Wang, Q. Ma, and Z. Cheng. Classifying malicious urls using gated recurrent neural networks. In *Proc. of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'18), Kunibiki Messe, Matsue, Japan*, volume 773 of *Advances in Intelligent Systems and Computing*, pages 385–394. Springer, Cham, June 2018.

Author Biography

Clayton Johnson is a senior undergraduate pursuing his Bachelor's in Computer Science from Colorado Mesa University (CMU) and earned his Professional Certificate in Cybersecurity from CMU in 2019. He is the former president of the CMU's Computer Science club and is a research fellow at the Cybersecurity Center at CMU. Clayton will begin pursing his Master's in Technology, Cybersecurity, and Policy at the University of Colorado Boulder in 2021.

Bishal Khadka is a senior undergraduate student pursuing his Bachelor's in Computer Science and Professional Certificate in Cybersecurity degrees at Colorado Mesa University (CMU). Bishal is currently the president of Cybersecurity club and a research fellow at the Cybersecurity Center at CMU.

Ram B. Basnet is an associate professor of Computer Science at Colorado Mesa University (CMU). He received his BS in Computer Science from CMU in 2004 and MS and PhD in Computer Science from New Mexico Tech in 2008 and 2012, respectively. His research interests are in the areas of information assurance, machine learning, and computer science pedagogy.

Tenzin Doleck received his PhD from McGill University in 2017. He is currently a post-doctoral fellow at the University of Southern California.