Cloud Computing and Data Science
http://ojs.wiserpub.com/index.php/CCDS/

UNIVERSAL WISER PUBLISHER

Research Article

Uniform Resource Locator Classification Using Classical Machine Learning & Deep Learning Techniques

Aws Rayyan, Mohammad Ghassan Aburas, Amjed Al-mousa*

Computer Engineering Department, Princess Sumaya University for Technology, Amman, Jordan
E-mail: a.almousa@psut.edu.jo

Received: 23 August 2022; **Revised:** 20 October 2022; **Accepted:** 24 October 2022

Abstract: In the Internet era, there is no doubt that the Internet has helped us in many ways by providing us with a means to communicate with anyone around the world. That is said, some people misuse such technology to conduct malicious behaviors. Many things could be exploited to perform such acts, but this work focuses on exploitation methods that use the uniform resource locator (URL). This paper presents the means to extract features from a raw URL. These are used to predict whether a URL is safe for a user to visit or not. The whole process of extracting the data and preparing it for a model is discussed thoroughly in this paper. Several machine learning (ML) models have been trained using different algorithms, including Catboost, RandomForest, and Decision trees, in addition to using and exploring several feedforward deep neural networks learning models. The best model achieved an accuracy of 95.61% on a test set using a deep learning model.

***Keywords*:** URL classification, feature extraction, random forest, machine learning, deep learning

1. Introduction

The Internet nowadays is growing faster than ever, and with this growth, cybercriminals emerged to conduct malicious behaviors for their gains. According to an Internet security report from Symantec [1], one in ten websites has malicious content. The attackers that make such websites do their work typically by creating a malicious URL and waiting for the victim to click on it. Once such a URL is clicked, the attacker can download a file to the victim's device (drive-by-download), which can do malicious things, such as displaying ads (adware), stealing information (spyware), or corrupting the victim's data. Another attack that uses URLs is phishing; it does not download anything malicious on the user's device. However, the website tricks the user into entering confidential information that the attacker can use later in another attack. With all of that in mind, there is an emerging need for a mechanism to identify the status of a website, whether it is benign or not, without visiting it, preventing a possible attack before it occurs. Therefore, by using ML, users will be proactively protected from many attacks.

In this paper, a typical machine learning strategy is followed, shown in Figure 1, where feature extraction and feature engineering are performed on the original dataset. These constitute part of this work contribution, as the feature selection and feature engineering help significantly in increasing the performance metrics. Then, the updated dataset is then split into a training dataset (80% of the data) and a testing dataset (20%). The training dataset is used to train both classical and deep machine learning models. Several classical machine-learning algorithms are explored to find the

DOI: https://doi.org/10.37256/ccds.4120231847

best model with the lowest generalization errors. Examples of such models are logistic regression, decision trees, and ensemble methods. The ensemble method was used in an attempt to enhance the performance. On the deep learning part, several models have been explored, each having a different number of layers and neurons. Also, advanced techniques that add some special layers to improve the model's performance are explored. The performance of the trained models is evaluated using the testing dataset. The performance is assessed using multiple performance metrics like accuracy, recall, precision, and F1-score.

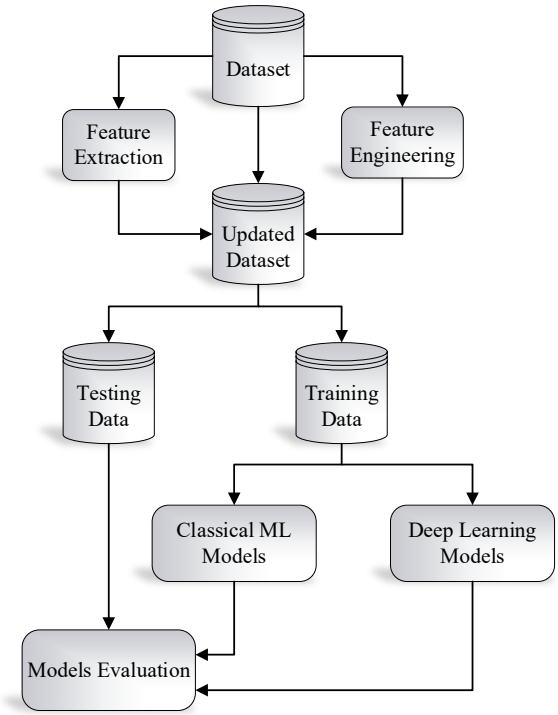

Figure 1. Overall Process Flow

Section 2 of this paper presents some work related to the URL classification problem; Section 3 gives an overview of the dataset that will be used to train the machine learning algorithms. It also presents all the methods utilized to extract useful features from the URLs in the dataset. Machine learning and deep learning models are discussed in section 4. The results of these models are presented in section 5. Finally, some of the methods used to enhance the model's performance are presented in section 6, along with a conclusion.

2. Related work

Due to the sophisticated ways and the rapid development of malware nowadays, it has become a relatively complex task for anti-malware software to keep up with this development. Therefore, such companies focus on preventing an attack before it occurs. When the source of the attack is web-based, an excellent way to detect and prevent an attack is by examining the URL of the website; hence many people have studied this area.

An approach based on blacklist technology to protect the user from malicious websites has been proposed in [2]. Such technology is widely used in many applications; however, when it comes to URL classification, it shows a lot of weaknesses. Due to the dynamic nature of malicious websites, as these websites usually change their URL periodically to overpass such technology, hence any URL that is not listed in such a database will go undetected. Thus, any ML-

based solution should be continuously retrained and updated with new incoming data to enhance its effectiveness. It is worth noting that machine learning is one of the modern approaches that are being used to solve such complicated problems in the medical field [3], education [4], and financial sector [5].

A convolutional neural network (CNN) model has been used to determine whether a website benign or not [6]. One of the significant advantages of this model is that it avoids the need for manual feature extraction as the usage of multilayer CNN automatically finds the hidden patterns in the input URLs, hence making the model efficient and independent of the feature engineering process. This model achieved an accuracy of 89% with one convolutional layer and 91% with two convolutional layers. In [7], several machine learning models such as logistic regression, support vector machine (SVM), k-nearest neighbors (kNN), and linear discriminant analysis (LDA) have been trained on this URL classification task. The paper compared the models based on the training time and many performance metrics such as accuracy, precision, recall, and f1 score. The paper showed that the KNN classifier obtained the highest accuracy on the test set at 93%. Further models such as long short-term memory (LSTM), full CNN, MLP, and majority voting of all these models are explored in [8]. The experimental results show that using one machine learning model (random forest) and two deep learning models (CNN and LSTM) in one voting ensemble reduces the overfitting that might appear when deep learning models are used individually. In addition, according to [9], URL classifiers would benefit more from a powerful feature extraction than that of making the neural network deeper.

3. Data preparation & analysis

This work aims to create a prediction system that can determine whether a website is benign or not. For this purpose, a huge dataset is built from smaller ones, such as ISCX-URL2016, Phishtank, and PhishStorm datasets.

3.1 *Dataset attribute information*

The dataset only has two columns, which are the URLs, and the types of the URLs. The type can be one of four for each website, which is 1-benign 2-phishing 3-defacement 4-malware.

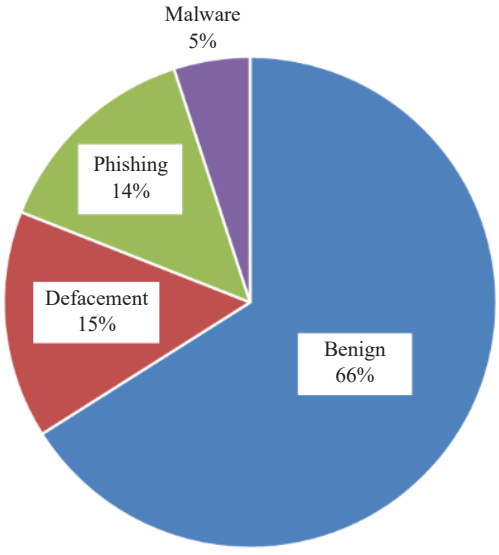

Figure 2. Labels Distribution

Figure 2 shows the distribution of the labels in the dataset. The labels of the dataset are not balanced; this must be considered when evaluating the model, as the accuracy metric will not be very useful in assessing the model. Instead,

the evaluation process must use metrics such as recall or precision.

Building a model using just a single feature (URL) is infeasible. Therefore, the first step is to extract some useful features from the URL text. In this paper, only lexical features will be considered. However, at the end of the article, some other features that can be extracted will be presented. These are expected to enhance the performance of the model significantly.

3.2 *Feature extraction*

A URL can be divided into 9-components; Figure 3 shows all the components a URL might have. The URL shown in the Figure is a special case where all components are shown. In the general case, five of such components are present in a URL because the domain part and the subdomain part are typically treated as one and are called the domain name, and the port is rarely present in a URL. The same thing applies to the query, as it typically results from user interaction, for example, searching for something in a search engine. In this paper, the six most relevant components of the URLs in the dataset will be extracted, namely:

- The protocol
- The domain name (subdomain + domain)
- The path
- The parameters
- The query
- The fragment

After extracting these components, it was found that only a few URLs in the dataset have the parameters, query, and fragment components. Therefore, these three components will not be considered in the analysis.

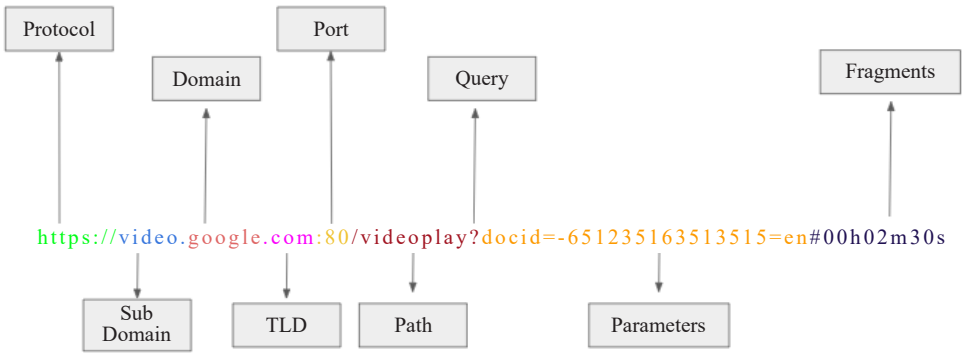

Figure 3. URL components

From each URL, several features will be extracted, such as the type of the protocol, whether it is HTTP, HTTPS, or FTP. This feature is of particular importance as there is a common misunderstanding by many people that the HTTPS protocol means that the website is safe to browse. Hence attackers exploit this mistake and use this protocol in their malicious websites. This feature is categorical; thus, it will be one-hot encoded. Other features such as the count of the top-level domain (TLD), the presence of an IP address or an E-mail, the presence of the words “server” or “client”, and the use of shortening services are also extracted. Other features extracted from the whole URL and each component are the length, digit count, number count, and the number of vowels inside the URL or the components.

One important attribute is the special reserved symbols inside the URL and its components. An example of such a symbol is the “@” symbol, which leads the browser to direct the user to the address provided after the symbol. So an attacker can inject the name of a legitimate URL before it to trick the user into visiting his malicious website. The following is a list of the reserved symbols used in the analysis [: / ? # [ ] @ ! $ & ' ( ) * + , ; =], other non-reserved

symbols that were extracted are [- . _ ~], finally these are some symbols which are considered un-safe [" < > % { } | \ ^ `] and must be encoded if it is to be used in a URL [10]. After extracting all these features, 110 numerical features are left that can be used in a machine learning model.

3.3 *Feature engineering*

After extracting all the features above, it has been noticed that 11 features do not exist in the respective component. Such as the count of the "[" symbol in the URL's path. Therefore, these features are dropped as they will not affect the model performance. Table 1 shows all the features that have been dropped. Furthermore, some features appear to be underrepresented. Hence such features are added together into a single feature with the constraint that the features of different components are not added together. Table 2 shows all the features that have been combined. After performing the mentioned operations, 63 distinct numerical features are identified.

Table 1. Dropped Features

URL Component	Dropped Features
Path	? # [ ]
Host	? # // , / [ ]

Table 2. Combined Features

URL Component	Combined Features
Full URL	> < ^ { } ( ) [ ] ! ; : ' ~
Path	> < ^ { } ( ) ! ; : ' ~
Host	> < ^ { } ( ) ! ; : ~
Protocol	HTTPS, FTP

As most of the algorithms that will be used to minimize the cost function are based on the gradient descent algorithm, the 63 features must be scaled first so that all the features end up with the same scales and the algorithm converges faster, furthermore choosing the proper method to scale the data, can significantly improve the performance of a model.

There are two standard methods to scale the features, either by using the min-max scaling, which makes the values of all the features between zero and one, equation (1) describes this method or by using the standardization, which is described in equation (2) which makes each feature has a mean of zero and a standard deviation of one.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

$$X_{stand} = \frac{X - mean(X)}{standard\ deviation(X)} \tag{2}$$

Both methods have been tried in the analysis, and it is observed that using the standardization technique to scale the data significantly improves the performance of the models, especially when using deep neural networks. The data is ready to be fed to a learning algorithm.

3.4 *Data pre-processing*

At this point, the prepared dataset is split into training and validation sets, with a distribution of 80% and 20% of the data in each set, respectively. The stratified splitting method ensures that each set has the same proportion of each label, which is essential when the classes are not balanced. A test set is crucial in evaluating the proposed model, as using it can ensure that the model will generalize well when exposed to new instances.

In addition, K-fold cross-validation is used with K = 10 to validate the results. K represents the number of sets to which a given dataset will be split. One of these sets will be taken as a test set, and all the other nine sets will be taken as training sets, so one would train the model on the nine sets and evaluate it using the test set. This process will be repeated four times until all the sets are used as a test set, then the evaluation result on each set is averaged. Figure 4 shows how this process works over four folds as a demonstration.

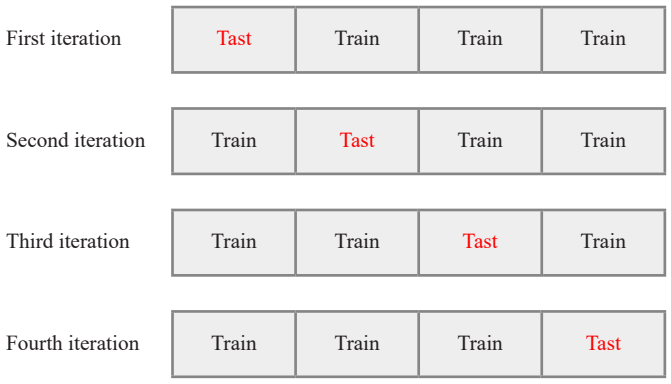

Figure 4. K-fold cross-validation process (Demonstration when K = 4)

It should be noted that using the K-fold cross-validation is computationally intensive. That's why it was used with classical ML models but not deep learning models, as it would have consumed a lot of resources. Hence, in the deep learning models, a fixed validation set was obtained out of the training set. The validation set size was 10% of the training set.

4. Algorithms

4.1 *Machine learning*

The ultimate goal here is to build a classifier that can determine which class a URL belongs to; therefore, several classifiers were tested to perform this task:

4.1.1 *Decision tree classifier*

The decision tree is a supervised machine learning algorithm mainly used with classification problems. Decision trees will keep splitting the data based on the features used in the dataset until reaching an outcome. Each node in the decision tree represents a feature, branches of the tree represent the decision rules, and each leaf represents the outcome.

4.1.2 *Random forests*

The random forest (RF) algorithm is a supervised machine learning algorithm, which is an ensemble learning method of decision trees. RF can be used for classification and regression problems by creating many decision trees on different random samples and then distinguishing between instances by checking the prediction outcome of these trees.

4.1.3 *Catboost classifier*

Catboost is a gradient boosting algorithm on decision trees. It is one of the most powerful and easy to implement gradient boosting algorithms in use. In addition, it implements symmetric trees, which helps decrease the prediction time.

4.1.4 *Light Gradient Boosting Machine (LGBM)*

LGBM is a gradient boosting framework that is based on tree learning algorithms. It uses Gradient-based One Side Sampling and exclusive feature building. LGBM uses GPU learning to provide faster and more efficient training; it can also handle large data with reduced memory consumption. LGBM can be used for many machine-learning problems such as classification, regression, and ranking.

4.1.5 *Ridge classifier*

The Ridge Classifier is based on the ridge regression method, which converts the data labels into values in the range [-1, 1]. Then it handles the machine learning problem as a regression problem. The prediction with the highest weight will be accepted as a target class. This paper explored other machine learning algorithms such as logistic regression, Adaboost with logistic regression as a base, and stochastic gradient descent, but all gave poor results.

4.2 *Deep neural network*

Overall, six different models were trained using deep neural network techniques; each model tries to modify the performance of its successor. It began with a relatively simple model and gradually increased the complexity of the model by changing the number of layers/neurons, changing the activation functions, and adding some specialized layers. Other techniques such as the early stopping and "learning rate schedule" have also been used. All deep learning models have an input layer of size 63, the number of features extracted, and the output layer will have four neurons, the number of the target label. The output layer uses the Softmax activation function, which is the most suitable type for this problem as the classes are mutually exclusive.

The first model (Model_1) trained has three hidden layers with 300, 200, and 100 neurons in each hidden layer, respectively. All the hidden layers will use the Rectified Linear Units (ReLU) as an activation function; the activation function is needed to add non-linearity to the model. Furthermore, the ReLU activation function helps increase the training speed of deep neural networks if it is used in all the hidden layers [11]. Finally, stochastic gradient descent (SGD) was used as an optimizer. Another model (Model_2) that is the same as this model but with two hidden layers instead of three is trained. However, the performance of the first model was better. So, it is safe to consider that three hidden layers are well suited for this problem. Hence three hidden layers were used for all of the other models.

The next model (Model_3) is similar to the first one, except that the SELU activation function will be used in all its hidden layers and the LeCun-normal weight initializer. The reason for this transition is explained thoroughly in the results and analysis section. In an attempt to improve the previous model's performance, three significant changes were implemented in the fourth model (Model_4):

• A batch normalization layer was added after every hidden layer, which helps in mitigating the vanishing/exploding gradient problem [12].

• A learning rate scheduler called the performance scheduler has been used.

• The Nadam optimizer, which Timothy Dozat introduced in [12], has been used.

As for the next model (Model_5), the last model was trained by adding two normal dropout layers, each having a drop rate of 20%. The dropout layer works by setting the input of a random subset of neurons with a percentage called

the drop rate to zero. By doing this, the neurons will work more independently, preventing the coadoption of neurons. It should be noted that in [13], it is recommended to use an alpha dropout layer instead of the normal dropout layer if you want to keep the self-normalizing property of the network. The alpha dropout layer keeps the mean and variance of inputs to their original values; therefore, it ensures the self-normalizing property even after the dropout. The final model (Model_6) implements such a layer. Table 3 provides a summary of all of the used deep learning models.

Table 3. Deep Learning Models Summary

Model name	Number Of Dense Layers	Number of Neurons per layer	Optimizer	Activation Function	Initializer	Early Stopping	Learning rate Scheduling	Batch Normalization	Dropout layer
Model_1	3	300-200-100	Stochastic Gradient Descent	ReLU	glorot	×	×	×	×
Model_2	2	200-100	Stochastic Gradient Descent	ReLU	glorot	×	×	×	×
Model_3	3	300-200-100	Stochastic Gradient Descent	SELU	Lecun normal	√	×	×	×
Model_4	3	300-200-100	Nadam	SELU	Lecun normal	√	√	√	×
Model_5	3	300-200-100	Nadam	SELU	Lecun normal	√	√	√	Normal dropout
Model_6	3	300 -200 -100	Nadam	SELU	Lecun normal	√	√	√	Alpha dropout

4.3 *Performance measurement metrics*

4.3.1 *Accuracy*

Accuracy is the ratio of the number of correct predictions by the machine learning algorithms to the total number of input instances. Equation (3) is used to find the value of this metric.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3}$$

4.3.2 *Precision*

Precision is a classification evaluation metric representing the ratio between the true positives and all the other positives. In this context, that would be the ratio between malicious URLs the model correctly identified and the total URLs identified as malicious. Equation (4) is used to find the value of this metric.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

4.3.3 *Recall*

The recall, also known as the true positive rate, is a classification evaluation metric representing the correctly identified true positives. In this paper, the recall will indicate many malicious URLs were correctly identified. Here a particular focus is given to this metric, as in this context, it is crucial in identifying malicious URLs, even if some benign URLs are identified as malicious. Thus, the mean recall of all the classes is mainly used to evaluate the model performance. Equation (5) is used to find the value of this metric.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

4.3.4 *F1-Score*

Since precision and recall are competing metrics, i.e. increasing one would make the other worse, the F1-score was defined as a metric that balances both precision and recall simultaneously. F1-score is defined as shown in Equation (6).

$$\text{F1-Score} = 2 \times \text{Precision} \times \text{Recall}/(\text{Precision} + \text{Recall}) \tag{6}$$

4.3.5 *False positive rate*

The False Positive Rate (FPR), which is commonly known as fall-out or false alarm ratio is the ratio between the negative classes that have been wrongly classified as positive and the total number of actual negative classes. This metric is important in the context of our paper as it shows the percentage of benign URLs wrongly classified as malicious. Having a low FPR means that the model rarely raises false alarms for a safe URL, which results in a better user experience. Equation (7) is used to find the value of this metric.

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

5. Results & analysis

5.1 *Machine learning*

The classifiers mentioned earlier have been tested, and Figure 5 shows their recall over the four folds of the cross-validation process. It can be seen from the figure that the random forest classifier and the CatBoost gives the highest performance, with the random forest performing slightly better than the Catboost. Therefore, the confusion matrix of these classifiers will only be considered in the analysis.

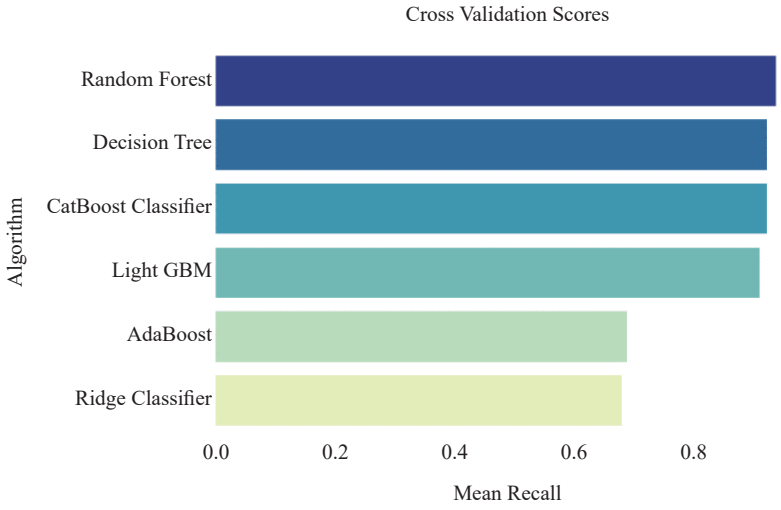

Figure 5. The recall for each machine learning classifier

Figure 6 and Figure 7 show the confusion matrices of the random forest and the Catboost classifiers after normalizing

them over the rows and removing the diagonal. This modification to the confusion matrix is needed to get better insights from the matrices. From the figures, it can be seen that most of the mistakes that both classifiers made in predicting the class of a URL are 0 (benign) when it is actually class 3 (malware). Adding more instances of type malware to the dataset can help overcome this problem since the malware class is underrepresented in the dataset.

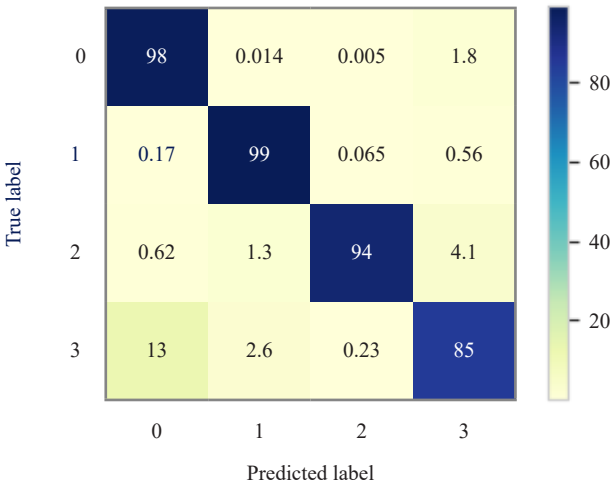

Figure 6. Random forest confusion matrix

The random forest classifier achieved a recall of 93.97% over the 10 folds of the cross-validation. It also achieved a 98.86% recall and 99.47% accuracy when evaluating it on the training set without cross-validation. This seems very promising; however, when the model was evaluated on the test set, it achieved a recall of 94.15% and an accuracy of 96.25%, which indicates that the model is overfitting the training set badly.

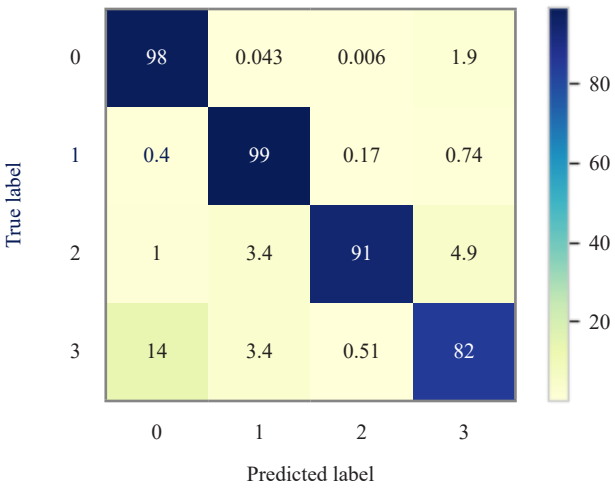

Figure 7. Catboost confusion matrix

The Catboost achieved a recall of 92.4% over the 10 folds of the cross-validation. It also achieved an accuracy

of 95.8% and a recall of 93.0% on the training set. These results are much lower than the scores of the random forest classifier. However, evaluating this model on the test set achieved an accuracy of 95.5% and a recall of 92.4%. Since the scores of the test and training set are comparable, one can see that the model is not overfitting as the random forest model. Table 4 shows the performance of the previous two algorithms.

Table 4. ML Algorithms Performance

Algorithm	Metric	Cross-validation	Training set	Test Set
Random Forest	Accuracy	96.2%	99.5%	96.3%
	Recall	94.0%	99.0%	94.2%
	Precision	95.6%	99.4%	95.7%
	F1-Score	94.8%	99.2%	94.9%
	FPR	1.95%	0.26%	1.89%
Catboost	Accuracy	95.5%	95.8%	95.5%
	Recall	92.4%	93.0%	92.4%
	Precision	94.6%	95.1%	94.6%
	F1-Score	93.4%	93.9%	93.5%
	FPR	2.24%	2.12%	2.22%

Since the random forest classifier performed better than the Catboost, it was attempted to find some hyperparameters for the model to decrease the overfitting problem discussed before. After searching, it appears that using 200 estimators and a max_depth of 25 gives the best performance with an accuracy of 98.26% and a recall of 97.20% on the training set. The scores on the test set are the same as those on the original random forest classifier with the default hyperparameters. However, this model reduced the overfitting problem slightly but did not eliminate it. Thus, additional hyperparameter tuning is needed.

5.2 *Deep neural network*

After training the first model (Model_1) for ten epochs, the obtained recall scores are 94.95% and 94.82% for the training and validation sets, respectively. The recall of the next model with two hidden layers Model_2 on the validation set is 94.42%. By inspecting the weights of the last hidden layer of Model_1, it has been noted that most of the weights are negative, indicating the dying ReLU problem [14]. This problem happens due to how the ReLU function works. The fact that the ReLU function always outputs a zero for any negative value results in a situation where neurons always output zero for any input if the weights of most of its connections are negative. In [15], several methods have been proposed to solve this problem. One way to avoid this problem was to use an alternative to the ReLU activation function, which is the Scaled Exponential Linear Unit (SELU) in all of the following models.

Furthermore, using the LeCun-normal as a weight initializer is recommended instead of the Glorot initializer. This adds the self-normalizing property to the network when the combination of the SELU activation function and such an initializer is used [13]. The recall score of the first model, which implemented all of these changes Model_3, is 95.82% and 95.51% on the train and validation sets, respectively. Model_4 recall on the validation set is slightly lower than the previous one (95.32%); however, the training time was lower. Specifically, the previous model took 39 minutes to train,

while this one needed 24 minutes only. Model_5 got a recall of 95.61% on the validation set, which is the best score. The final model (Model_6) got a recall of 94.38%, which is lower than the previous two models; hence, adding the alpha dropout layer was not useful.

Figure 8 shows the performance of all the deep learning models using the performance metrics discussed before. It can be seen that Model_5 has the best performance on the validation set in all metrics compared to the other models.

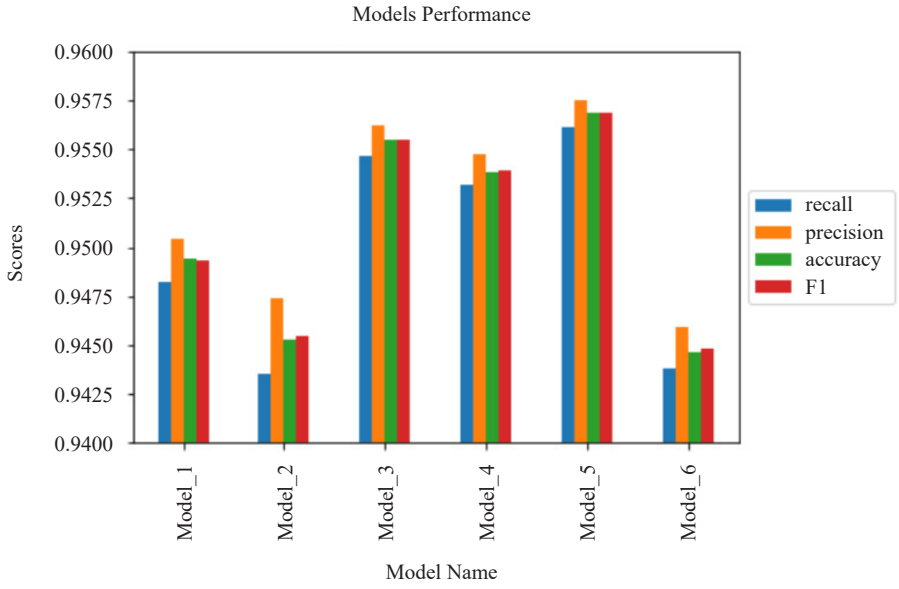

Figure 8. Deep learning models scores on the validation set

Since Model_5 has the best performance, it might be helpful to look at how the recall score changed with every epoch during training in the validation and training sets. A graph showing that is presented in Figure 9.

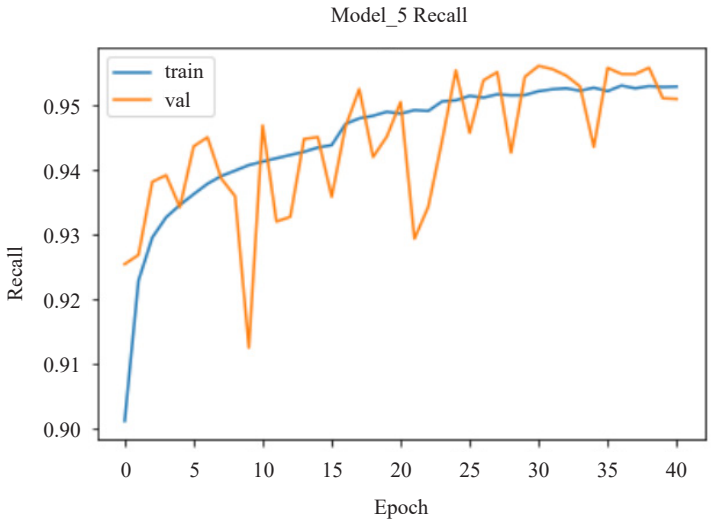

Figure 9. Model_5 recall score on each epoch

In Figure 9, it can be seen that the training set score seems to be less than that of the validation set. This difference is due to the use of the dropout layer in the model, so the training set recall score should be shifted up a little bit to be representative. The validation set recall is too volatile, indicating an overfitting problem. It can be seen that between the 29th and 35th epochs, the curve stabilizes, and since the early stopping technique was used, the final model selected was the one in the 30th epoch, which did not suffer from the overfitting problem.

Finally, after evaluating Model_5 performance on the test, the model achieved good results with an accuracy of 95.6% and a recall of 95.5%. Table 5 shows the accuracy, recall, Precision, and F1-Score of the model on the different sets. It is observed that the scores on the training set are better than that of the other two sets; however, it is slightly higher, which is normal. Thus, the models are expected to generalize well on new URLs. The precision of the model is higher than the recall. So, suppose the model is deployed in production. In that case, it will still give the users a good experience without interrupting them with fake alerts about security issues when browsing a safe website.

The F1 score, the harmonic mean of the precision and recall, is also high in the model, making it suitable for deployment in production as it will effectively catch malicious websites while keeping the false positive alerts low.

Table 5. Model_5 Performance Metric across All datasets

Metric	Training set	Validation set	Test Set
Accuracy	95.9%	95.7%	95.6%
Recall	95.9%	95.6%	95.5%
Precision	96.0%	95.8%	95.7%
F1-Score	95.9%	95.7%	95.6%

6. Conclusion

This paper presented several machine learning and deep learning models that can be used to determine whether a URL is malicious or not using the URL lexical features only. The performance of the models can be significantly enhanced by adding other features such as features related to the domain, including the domain age, the domain expiration date, the country of the domain, and the existence of an entry for this URL in the WHOIS database. Other features that can also be added are features related to the website itself. This includes the color of the website background, the customization of the status bar, usage of website forwarding, and the usage of iFrame redirection. Natural language processing (NLP) models can also be used in this context, especially in detecting phishing websites. In this paper, these features were not used due to the excessive network requests needed to extract such features. Hence a considerable amount of time is required to extract such features. In addition, future work can include the use of advanced architectures like CNN and LSTM, and compare their performance to existing models.

Conflicts of interest

The authors declare no competing financial interest.

References

[1] O'Gorman B, Wueest C, O'Brien D, Cleary G, Lau H, Power JP, et al. *Internet Security Threat Report*. Symantec; 2019. Available from: https://docs.broadcom.com/doc/istr-24-2019-en.

[2] Akiyama M, Yagi T, Hariu T. Improved blacklisting: Inspecting the structural neighborhood of malicious URLs. *IT Professional.* 2013; 15(4): 50-56. Available from: doi: 10.1109/MITP.2012.118.
[3] Abdulhadi N, Al-Mousa A. Diabetes detection using machine learning classification methods. *2021 International Conference on Information Technology (ICIT)*. IEEE, Amman, Jordan; 2021. p.350-354. Available from: doi: 10.1109/ICIT52682.2021.9491788.
[4] Bitar Z, Al-Mousa A. Prediction of graduate admission using multiple supervised machine learning models. *2020 SoutheastCon*. IEEE, Raleigh, NC, USA; 2020. p.1-6. Available from: doi: 10.1109/SoutheastCon44009.2020.9249747.
[5] Atwah A, Al-Mousa A. Car accident severity classification using machine learning. *USA 2021: International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT).* IEEE, Zallaq, Bahrain; 2021. p.186-192. Available from: doi: 10.1109/3ICT53449.2021.9581646.
[6] Singh A, Roy PK. Malicious URL detection using multilayer CNN. *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE, Zallaq, Bahrain; 2021. p.340-345. Available from: doi: 10.1109/3ICT53449.2021.9581880.
[7] Chiramdasu R, Srivastava G, Bhattacharya S, Reddy PK, Gadekallu TR. Malicious URL detection using logistic regression. *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*. IEEE, Barcelona, Spain; 2021. p.1-6. Available from: doi: 10.1109/COINS51742.2021.9524269.
[8] Rakotoasimbahoaka AC, Randria I, Razafindrakoto NR. Malicious URL detection using majority vote method with machine learning and deep learning models. *2020 International Conference on Interdisciplinary Cyber Physical Systems (ICPS)*. IEEE, Chennai, India; 2020. p.37-43. Available from: doi: 10.1109/ICPS51508.2020.00013.
[9] Skalický M. *Classification of URLs using deep neural networks*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.
[10] Berners-Lee T, Fielding R, Masinter L. *Uniform resource identifier (URI): Generic syntax*. The Internet Society; 2005. Available from: https://datatracker.ietf.org/doc/html/rfc3986.
[11] Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. Fort Lauderdale, FL, USA; 2011. p.315-323.
[12] Dozat T. *Incorporating nesterov momentum into adam*. ICLR 2016 workshop paper; 2016. Available from: https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.
[13] Ioffe S, Szegedy C. Batch normalization: *Accelerating deep network training by reducing internal covariate shift*. In *Proceedings of the 32 nd International Conference on Machine Learning.* JMLR Workshop and Conference Proceedings; 2015. p.448-456.
[14] Lu L, Shin Y, Su Y, Em Karniadakis G. Dying relu and initialization: Theory and numerical examples. *Commun. Comput. Phys*. 2020; 28: 1671-1706. Available from: doi: 10.4208/cicp.OA-2020-0165.
[15] Klambauer G, Unterthiner T, Mayr A, Hochreiter S. Self-normalizing neural networks. *31st Conference on Neural Information Processing Systems (NIPS 2017)*. Long Beach, CA, USA; 2017.