Journal of Network and Systems Management (2021) 29:21
https://doi.org/10.1007/s10922-021-09587-8

URLdeepDetect: A Deep Learning Approach for Detecting Malicious URLs Using Semantic Vector Models

Sara Afzal[1] · Muhammad Asim[1] · Abdul Rehman Javed[2] · Mirza Omer Beg[3] · Thar Baker[4]

Received: 1 June 2020 / Revised: 17 January 2021 / Accepted: 3 February 2021

Abstract
Malicious Uniform Resource Locators (URLs) embedded in emails or Twitter posts have been used as weapons for luring susceptible Internet users into executing malicious content leading to compromised systems, scams, and a multitude of cyber-attacks. These attacks can potentially might cause damages ranging from fraud to massive data breaches resulting in huge financial losses. This paper proposes a hybrid deep-learning approach named *URLdeepDetect* for time-of-click URL analysis and classification to detect malicious URLs. *URLdeepDetect* analyzes semantic and lexical features of a URL by applying various techniques, including semantic vector models and URL encryption to determine a given URL as either malicious or benign. *URLdeepDetect* uses supervised and unsupervised mechanisms in the form of LSTM (Long Short-Term Memory) and *k*-means clustering for URL classification. *URLdeepDetect* achieves accuracy of 98.3% and 99.7% with LSTM and *k*-means clustering, respectively.

Keywords Malicious URL detection · Security and privacy · Word embedding · Deep neural networks

1 Introduction

An Online Social Network (OSN) is a platform where people interact and communicate with each other for various purposes such as entertainment, marketing, and business. Users share their ideas, photos, videos, events, and personal information in the form of posts on these platforms [1, 2]. Posting all of these details on online social platforms not only creates opportunities for people to exploit the users [3], but the simple use of any such platform can also lead to a lot of privacy concerns [4–6]. People may give out too much of their personal information, and the sites may not

✉ Muhammad Asim
muhammad.asim@nu.edu.pk

Extended author information available on the last page of the article

Published online: 04 March 2021

take adequate steps to protect their privacy. This creates an opportunity for cybercriminals to attack their privacy and get what they want [7–9]. This is where phishing attacks get very common. Such attacks are done by following a common method of shortening URLs through URL shortening services [10] and adding them in different posts and statuses on social media. A social media phishing attack involves deceiving the user to click on an infected link which can lead to a drive-by download attack [11, 12] or can steal the user's credentials either from the site or by asking the user indirectly to enter their details [13]. Most of the URLs look very similar to the original URLs tricking the users into believing that they are opening the actual link [14].

Among many popular social media platforms, Twitter is a very famous social network with the majority of most celebrities, politicians, and businessmen using it. It has a restriction of using only 140 characters in a post hence it is famous for including shortened URLs to provide additional details along with the post. This provides an opportunity for cybercriminals to take advantage of shortened URLs and harm the user's system or get access to their private data [15]. Drive-by download attacks and luring the users into providing private information are considered as growing dilemmas of cyber-attacks especially when cybercriminals target large-scale trending events on different online social platforms [16]. The trending topics and their associated terms or hashtags are the easiest approaches for cybercriminals to lure information-seeking users [17]. URLs are added along with the post and these URLs consist of comprise of additional information, evidence to the topic mentioned in the post, or an image to support that topic.

Attacks such as drive-by download attacks can be prevented by studying the behavioral analysis of malware which leads to the identification of malicious URLs [16, 18]. Researchers have emphasized the detection of malicious and benign URLs and worked on building machine learning models for the classification of URLs. The classification works on different kinds of features which include machine activity features [16], tweet features [19], URL based features [14, 20], lexical features [10], etc. These features help in detecting whether a particular URL is harmful to a user and some works [21] include protecting the users from clicking those URLs. It has become very important to build such a mechanism that provides good accuracy in detecting malicious URLs to protect the private data [22] and be safe from monetary losses. Figure 1 demonstrates how accessing an unknown URL can lead to harming the user's system if there is no way adopted to check whether it is malicious or not. However, if the URL is checked before opening it, the user can be saved from any potential damage [23].

The motivation behind this paper is the detection of malicious URLs considering Twitter as the main aspect to protect its users since Twitter is the most used platform with 330 million[1] monthly active users today. This makes it very important to protect Twitter from cybercriminals as they can target a large number of audiences to harm them via Twitter. The existing work of researchers is mostly based upon features that can be collected only after clicking the URL [24, 25] such as

[1] https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/.

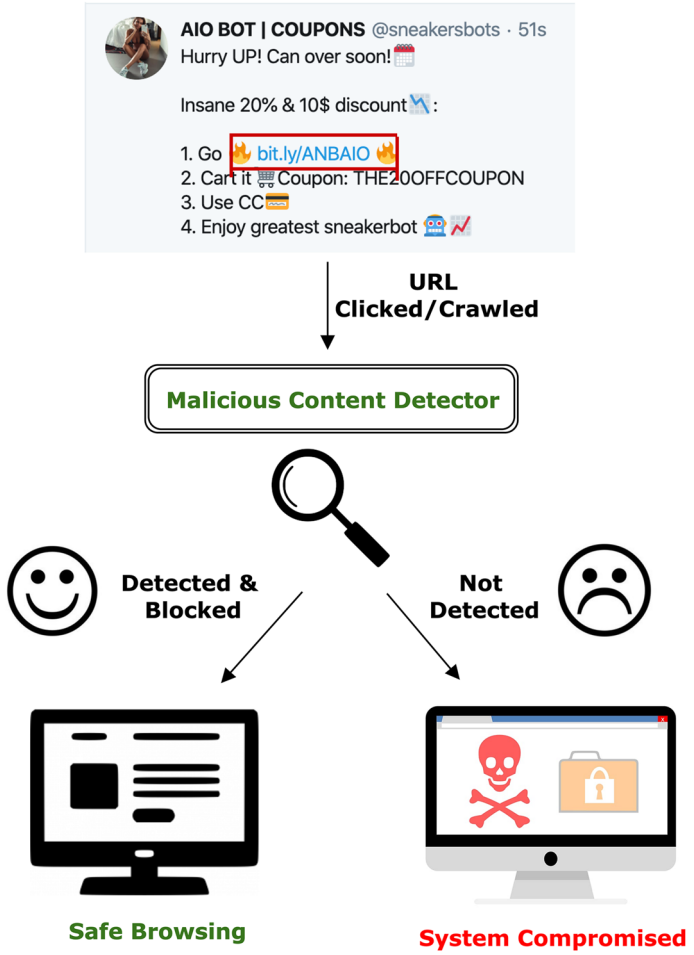


Fig. 1 Consequences of detection and non-detection of malicious URLs

machine activity features. Working with those features may harm the user's system before any malicious activity can be detected and blocked. Since those features require the URL to be clicked on, causing harm to the user or the system, therefore, it is not a sufficient method to consider. Other features are based on tweet attributes [26] that are difficult to get at run-time during the implementation of a prediction model, hence making the algorithm very complex. Therefore, this leads to the need for developing a security management system to detect malicious URLs, which can work in all kinds of scenarios without harming the system or increasing the complexity of the algorithm.

This paper introduces a novel approach named *URLdeepDetect* in the field of cybersecurity management for detecting malicious URLs by implementing and demonstrating work on two different techniques. The approach includes working

Table 1 Table of abbreviations

Abbreviations	Explanation
LSTM	Long Short-Term Memory
URL	Uniform Resource Locator
NLP	Natural Language Processing
DNS	Domain Name System
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
SVM	Support Vector Machine
MLP	Multilayer Perceptron
NB	Naïve Bayes
RF	Random Forest
CBOW	Continuous Bag-of-Word
TF-IDF	Term Frequency-Inverse Document Frequency
NLTK	Natural Language Toolkit

with semantic vector models of URL tokens, along with URL encryption. It also includes applying a modified version of recurrent neural network (i.e. LSTM), and *k*-means clustering as the classification models. The first technique uses tokenization and word embedding and classifies the URLs via LSTM. The second technique uses URL encryption and classifies via *k*-means clustering. This work makes the following The key contributions of this work are as follows.

- A hybrid deep-learning approach named *URLdeepDetect* for time-of-click URL analysis and classification to detect malicious URLs.
- The use of word embedding generated by a vector space model is made to capture the semantics of URL tokens and then feed them into an LSTM to capture the sequential semantics of the entire URL.
- A novel URL encoding mechanism is developed and presented and is used in combination with vector embedding to employ clustering for unsupervised URL classification.
- A thorough empirical analysis of our techniques is presented and compared the results with highly tuned conventional machine learning models.

The remainder of the paper is structured as follows. Section 2 discusses the problem formalization of the paper and explains the lexical features of the URL. Section 3 presents the related work. Section 4 introduces the URLdeepDetect approach. The experimental setup, results, and comparative analysis are articulated in Section 5. Finally, section 6 concludes the work. Table 1 denotes the abbreviations used in this work.

2 Detection of Malicious URLs

2.1 Problem Formulation

The primary focus of this paper is to detect malicious URLs on different social platforms. To solve such a problem, a binary classification [27] is used since the target classes contains two labels i.e. malicious or benign. Consider a dataset containing URLs and labels (i.e. malicious and benign). The URLs can be represented as $\{(u_1, y_1), \ldots, (u_T, y_T)\}$ where u_1 to u_t are the URLs whereas y_1 to y_t are the labels of each URL respectively. The variable y can belong to the set $\{-1, +1\}$ since it denotes the label of the URL. We can consider $y = +1$ to be a benign URL while $y = -1$ to be a malicious URL. This means that if the label corresponding to a particular URL depicts the URL as benign, the value of y will be $+1$ and if the label depicts the URL as malicious, the value of y will be -1. The process of classification involves two steps that are followed in the approach. The first one is to attain a feature demonstration $u_t \rightarrow x_t$. Here x_t belongs to $\mathbb{R}^n$ which is a feature vector with the dimension n and is used to represent a URL u_t. A function denoted by $f : \mathbb{R}^n \rightarrow \mathbb{R}$ for prediction must be learned which is basically the second step of the process. The purpose of this function is the prediction of a score so that a URL could be placed into any of the two classes. The function makes the prediction which is represented as $\hat{y}_t = sign(f(x_t))$. We perform these steps to minimize the loss function which is basically about trying to reduce the total number of mistakes in the complete dataset. For our problem, f represents LSTM and k-means clustering which are the classification models that we used for the detection of the URLs. Later, the use of several evaluation metrics is done to evaluate our prediction function's performance which in our case includes accuracy, precision, recall, and f-measure.

2.2 Lexical Features of URL

With the help of the lexical features of a URL [28], a raw URL is represented by u and is transformed into a vector of features represented by x. To do that, the URL is split into different tokens. The splitting takes place on special characters in the URL [29]. A huge number of tokens are obtained after splitting the URLs of the entire dataset. A dictionary is made in which all unique tokens are placed and each feature then represents a word which is represented by w_i. In one of the two techniques, with the provision of unique features represented by M, every given URL represented by u_t is worked upon for mapping it to a vector represented by x_t which belongs to $\mathbb{R}^M$. This is done in such a way that the i_{th} component in x_t is mentioned as 0 stating that the word w_i is not present in the URL and it is mentioned as 1 stating that the word w_i is present in the URL. The second technique involves URL encryption instead of working with a vector model.

Each URL splits up into many tokens, hence creating a very large dataset with a huge feature size. The new dataset can even be greater in size than the original one, since there are more than one tokens for every URL. Next, the model is to be trained again since we have a new dataset. For training again, the size grows for

both the model as well as and features. The existing models, by using these features, have some limitations. Firstly, the sequence of the tokens is not maintained once the URL is split and the unique tokens are added to the dictionary. Secondly, the words which appear rarely or just once in the dictionary do not provide meaningful information. Lastly, new words in test data cannot be interpreted properly since they had not appeared in the data used for training purposes. Hence, there is no success in working with the models that fail to find any meaningful information behind such words. To overcome these limitations, both our models play their part and give high-quality results.

3 Related Work

Many researchers have worked on the detection of the nature of URLs as either malicious or benign [30–32]. In this section, we describe the work presented by those authors along with the machine and deep learning [33, 34] models that they have used. The division of The related work section has been split can be made into 3 sub-sections based upon feature categorization. The first sub-section covers the papers that use tweet features and machine activity data to predict the URLs as malicious or benign. The second sub-section includes those references that use lexical, domain/host-based, and URL based features to achieve accuracy in prediction. The last sub-section consists of those papers that discuss vector models and neural networks for phishing detection. The literature review is done in the respective sections.

3.1 Usage of Tweet Features and Machine Activity Data

In [16], the authors focused on achieving four targets. Firstly, they developed a system for the classification of URLs to determine them as either malicious or benign and they did this in seconds after the URL was clicked by the user. Secondly, the model that was learned was examined and interpreted to study the relationship between machine activity and malware presented by drive-by-downloads on Twitter. Thirdly, their paper included work for the identification of such features that could generalize across different happenings. Lastly, the authors tried to understand how much data would be sufficient for this problem. The limitation in their work is that they have used only it uses machine activity features and that too is very limited , only. As for [21], improvements have been made to previous work [16] as the authors worked on building a machine learning model with the use of different features such as machine activity features along with the features of a specific tweet to block and prevent a drive-by download attack. This approach allowed them to identify the malicious URLs before these URLs harm the user or the system after their execution and then take time to recover from such harm. The authors in [19] claimed that the approach of using only visible features of tweets and user profiles to detect malicious URLs is practicable and gives good enough results. Tweet features are extracted using python scripts and those features are divided into three categories including content, context, and social features. They claimed that if

machine activity features are not used, their predictive model can still give the best results. The authors in [35] focused on developing a detection system for the nature of URLs. The main idea behind the paper was to explore the relationship between different URLs and their re-directions to different pages. Therefore, they used two types of features. The first type consisted of features extracted from the idea of URL re-directions whereas the second type involved features that had data drawn out from tweets. The work in [36] solved the problem of malicious URL detection into two parts. The first part involved the training of the dataset to detect the maliciousness of URLs while the second part consisted of the work that they proposed i.e. building a mechanism for security alerts. A total of 11 features were used for training. The authors used Random Forest (RF) and achieved an accuracy of 97.50%.

3.2 Usage of Lexical, Domain/Host, and URL Based Features

Classification of URLs was done using a machine learning approach in [10]. The authors extracted lexical features from a URL and assumed that the features for both malicious and benign URLs are different which helps in classifying them. The deployment of the model was done in the FireEye Advanced URL Detection Engine (FAUDE), in which malicious URLs were detected in a large number. The authors in [37] claimed that all the authors who have worked in this domain have focused on binary classification and none of them have done multi-class classification to detect the nature of URLs. According to them, the URLs are only classified as malicious or benign and not as any other kind of spam. Therefore, the main idea behind their paper was to introduce a multi-class classification system so that the nature of URLs could be identified more specifically. For training purposes, they used a supervised machine learning approach. In [38], authors analyzed approaches that extract features from phishing websites and train classification models with an extracted feature set to classify phishing websites. They created an exhaustive list of all features used in these approaches and categorized them into 6 broader categories and 33 finer categories. They extracted features from the URL, URL redirects, hosting domain (WHOIS and DNS records), and popularity of the website and analyzed their robustness in classifying a phishing website. Their emphasis was on determining the predictive performance of robust features. Later, they evaluated the classification accuracy when the entire feature set was used and when URL features or site popularity features are excluded from the feature set.

3.3 Usage of Vector Models and Neural Networks

The authors in [39] presented present a methodology in which they made use of lexical features of URLs to classify their maliciousness using machine learning techniques. The methodology consisted of using the techniques of Natural Language Processing (NLP) which involves working with word-level vectors and n-gram models. With the help of this technique, the authors classified the URLs as either malicious or benign. They achieved an accuracy of 97.1% with a machine learning model named SVM (Support Vector Machine). The authors in [40] proposed an approach

of phishing detection using the techniques of deep learning. The mapping of the URLs on websites and their characters was done to different documents and words respectively with the use of a word embedding technique called word2vec. Furthermore, the combination of embedding of characters was done with URL structures to acquire the URL in the form of vectors. To achieve this, the authors split the URL into five components. The work in [41] achieved the purpose of classifying the URLs as malicious or benign using CNNs (Convolutional Neural Networks) along with the use of word2vec model for word embedding. These techniques were used to classify news articles and tweets. Different embedding techniques were used by the authors which include word2vec, Continuous Bag-of-Word (CBOW), and Skip-gram. The authors built the neural network with two of these algorithms which are CBOW and Skip-gram. The accuracy of the classification of the neural network was measured using the above mentioned two techniques and achieved the highest accuracy with CBOW Model. Authors in [42–45] presented studies to detect intrusion attacks in a connected system. The authors in [46] conducted extensive experiments on large-scale datasets and used deep learning-based solutions for detecting the maliciousness of URLs. Their method involved breaking the URL into words and characters and then feeding them all to the convolutional neural network. The technique of identifying unique characters and unique words first and then converting them into matrix representations later was used. In particular, two CNN models were made with the major difference of character level and word level embedding in each model. Both these models were combined to optimize the final model for prediction.

The research papers discussed above show that a variety of machine learning algorithms have been used to solve the problem that has been encountered i.e. detecting the URLs as malicious or benign. The common techniques that have been used extensively in the majority of the work are Random Forest, Naïve Bayes, Multilayer Perceptron, Decision Trees, Logistic Regression, Support Vector Machine, Bayes Net, and Neural Network. Most of the papers have used these algorithms on the datasets of features which are tweet based, machine activity-based, or URL based making all the papers similar to one another in some way. Therefore, we discuss a novel approach in this paper which no other work has yet introduced i.e. make use of two techniques which include word embedding and URL encryption for the detection of the nature of URLs. We also implement a few techniques mentioned above from our related work to compare our results with them.

4 URLdeepDetect

In this section, we present our proposed approach URLdeepDetect, which consists of two different techniques. We introduce a novel generalized approach that not only detects Twitter's malicious URLs but URLs on all other platforms. In the implementation, first, we pre-process the dataset and then we tokenize the URLs. We apply word-level embedding in the first technique whereas URL encryption in the second technique. In the process of word embedding, we calculate the semantic similarity

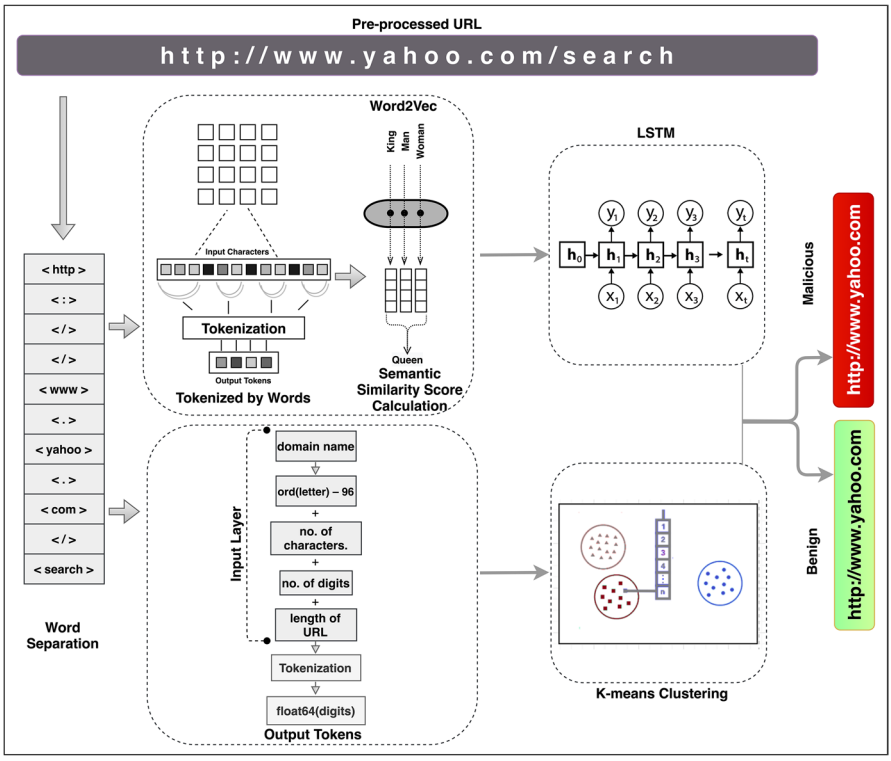


Fig. 2 Overview of the proposed architecture

score and then make vectors. Lastly, we perform classification using LSTM and *k*-means clustering.

The pseudo-code of our algorithm and the processing flow is illustrated in Algorithm 1. The architecture of the proposed approach is shown in Fig. 2. It demonstrates the methodology of the proposed model. The pre-processed URL is taken and two different techniques are applied to it. The upper half of the diagram displays that tokenization and Word2Vec are performed on the URL and later LSTM is applied on the vector to get results. The lower half of the diagram demonstrates the second technique where URL Encryption is performed and later *k*-means clustering is used to get the results.

Algorithm 1 Algorithm for Malicious URL Classification

Input: Sequence of Raw URLs;
Evaluation Measure: Accuracy, Precision, Recall, F-Score, Loss, ROC Curve, Confusion Matrix;
Output: Malicious or Benign;

```
Split_URL = [] %Create the Mapping Word to the Index Array;
translated Words = null;
for data in URL_dataset: do
    Split_URL_dataset= url[data].split('.');
end for %Return Splitted URL dataset
Initialize Train (80% of URL dataset) and Test Data (20% of URL dataset);
Train and Validate (Training_Data, Testing_Data);
Reshaping Vector;
Initialize Word2Vec for Training_Data (min_count, size, window, sg); % Make Training Model Vector
Initialize Word2Vec for Testing_Data (min_count, size, window, sg); % Make Test Model Vector
Initialize Pretrained Weights for Training Data
Vocab_size and embedding_size = Pretrained Training Weights
Initialize Pretrained Weights for Test Data
Vocab_size_test and embedding_size_test = Pretrained Testing Weights
for sublist in range (len(X_train) : do
    for val in range (len(X_train[sublist])) : do
        Convert TrainingWord-to-TrainingIndex
    end for
end for %Return Converted Training dataset
for sublist in range (len(X_test) : do
    for val in range (len(X_test[sublist])) : do
        Convert TestWord-to-TestIndex
    end for
end for  %Return Converted Test dataset
col=len(max(Returneddataset,key=len)) %Defining Max Length
i = [X_train, X_test, Y_train, Y_test]
for data in i: do
    i = np.zeros([len(i), col], dtype=np.int32)
end for  % 2d list definition with 0's of nxn shape
Initialize Sequential Model;
Input the embedding size vector to the first embedding layer;
Add Dropout;
Feedforward to the 2nd LSTM layer;
Add Dropout;
Feedforward to the Dense fully-connected layer with 4 hidden units;
Feedforward to the 2nd Dense Layer;
Use sigmoid for output;
Use "binary cross entropy" as loss function and "adam" optimizer;
for epoch in range (10): do
    Evaluate Loss, Validation Loss
    Evaluate Accuracy and Validation Accuracy
    Evaluate Precision, Recall, F-Score, Roc Curve and Confusion Matrix
end for %Return Output
```

4.1 Technique 1: Word Embedding and Long Short-Term Memory

4.1.1 Tokenization

The first step of our implementation includes the process of tokenization of URLs, which is a necessary step of Natural Language Processing (NLP) tasks. Before

performing any further steps on the data, it is a mandatory step to split the text into minimal meaningful units. Line 1 to 5 of Algorithm 1 demonstrates the code for the process of tokenization. We use the basic tokenizer, which is present in NLTK (The Natural Language Toolkit) and is basically supported in python to work with programs that involve techniques related to human language data. NLTK consists of a lot of libraries that can be used on strings and characters to find semantic meanings behind them. Among those libraries, we use the tokenize library to parse the URL into different tokens. We make an array to store the tokens after parsing. The tokens of each URL are appended one by one in the array which is later passed on to the vector model discussed in the next section.

Our tokenizer works in a way that it splits up the URL into different words by using an unsupervised algorithm to build a vector model that can be given as an input to LSTM. We use two modules of the tokenize library which include word_tokenize and sent_tokenize. The first module i.e. word_tokenize is used to split words in the URL. It returns the syllables from a word where a single word may contain two or three syllables. The second module i.e. sent_tokenize function is a submodule of word_tokenize.

4.1.2 Word Embedding: Word2Vec

The next step involved in the process is to create a vector model using the tokens that we obtain in the previous step through the process of tokenization. We do so by following the method of vectorization, also known as word embedding. It is the process of converting words into numbers. Therefore, we use it for the same purpose i.e. we try to find meaning out of the words and strings by actually converting the words into numbers which are basically vectors. Each word is mapped to its corresponding vector. An easy way to understand how arithmetic operations can be performed on words is with an example. Let us consider a famous example: $king - man + woman = queen$, which is depicted in Fig. 2 of the proposed approach. Here, if we add the vectors which are associated with the words *king* and *woman* and then minus the vector associated with the word *man* will give us a vector associated with *queen*. Therefore, after the conversion of words to vectors, we use some techniques to find the similarity among the words. Line 9 to 29 in Algorithm 1 demonstrates the process of vectorization.

Many approaches to perform word embedding are being used with different pros and cons. These approaches include BOW (Bag of Words),[2] TF-IDF (Term Frequency – Inverse Document Frequency) Scheme[3] and Word2Vec. The technique that we use for word embedding is Word2Vec. In this method, the use of neural networks is made to do the conversion of words into vectors such that the words which have their semantic score similar to each other are considered to be closer in a space

[2] https://towardsdatascience.com/machine-learning-text-processing-1d5a2d638958.

[3] https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089.

of dimension n. We import the word2vec model from the Gensim library and then use it to build word embeddings. We train the word2vec model by making use of an algorithm named skip-gram. To create a word2vec model, we create a corpus first and then pre-process the content. For the training of the model, we use nltk. sent_tokenize and nltk.word_tokenize utilities. We use the class word2vec when the array is passed to it keeping in mind that the array consists of all the words. This completes the creation of our word2vec model that converts the words into their corresponding vectors. The vocabulary size does not have any effect on the vectors that we have built.

4.1.3 Long Short-Term Memory (LSTM)

The final step of the first technique includes the details regarding the model that we implement to predict the accuracy of the data given. Hence, the final step in this technique is to feed the vectors into a neural network. We use LSTM as our classification model which is a tweaked version of the recurrent neural network. Using LSTM as the classification model gives us the biggest advantage since we are dealing with word embeddings. The model needs to learn the embeddings which can be done best while fitting the LSTM. Traditional models have a limitation of having no memory. Recurrent Neural Networks like LSTM can be used to tackle this problem. Since we are dealing with Natural Language Processing techniques in our work, LSTM can work best by remembering every input and making it easier to generate meanings out of words. LSTMs can make little modifications by multiplications and additions depending upon the need as they can selectively remember or forget things. After the creation of vectors, we work on feeding those vectors to the LSTM for training. Line 30 to 38 of Algorithm 1 demonstrates the process of training the model through LSTM. We prepare the data in such a format that for the prediction of 'E' in 'BLUE', the input given is the first three letters in an array and the outcome of that input is the last letter. To define the LSTM model, we use the sequential model. The first layer is embedding which takes the vocabulary size as input dimension, embedding size as output dimension, and the pre-trained weights as the weights. We then add an LSTM layer to make sure that the problem of receiving data in a scattered manner by the next LSTM layer can be avoided. The model also has to be built in such a way that there is no over-fitting in it. Hence, we use the dropout layer for this purpose after every LSTM layer. Next, we use a fully connected layer i.e. dense layer and the activation function which we use with it is 'relu'. Then, we apply another dense layer with 1 unit. Lastly, we use the Sigmoid function the output of which is basically a vector. The vector has values 0 and 1 where 0 depicts that the value should be forgotten whereas 1 means that the value should be remembered.The complexity analysis is depicted via a graph is represented in Fig. 3 in which the running time of each epoch is shown and the details of the used parameters are demonstrated in Fig. 4. We then compile our model by setting the loss to binary_crossentropy. The optimizer is set as Adam and the metrics are set to accuracy. After that, we fit our model and run 10 epochs on it for

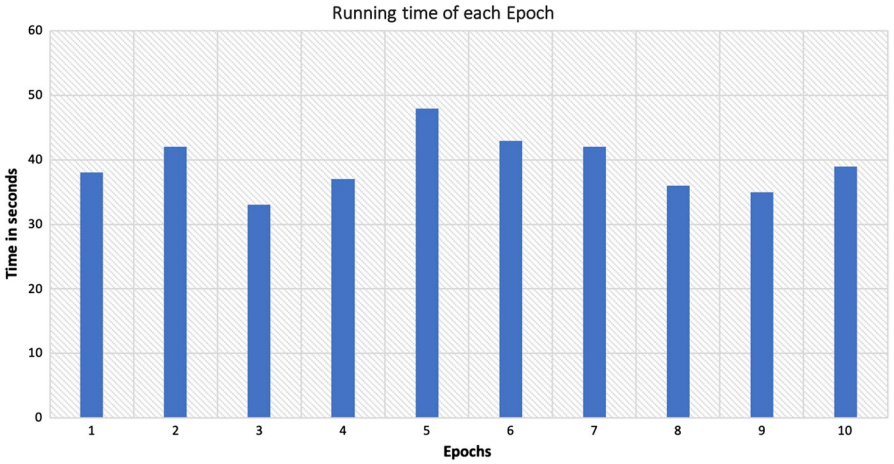


Fig. 3 Complexity analysis of LSTM model

training which gives us an accuracy of 98.3%. During model fitting, we pass the parameter of *validation_split* setting it to 0.1 which also makes sure that 10% of the training data is used for validation. With that, we also obtain the validation loss and accuracy.

4.2 Technique 2: URL Encryption and k-Means Clustering

This approach involves working with the technique of *k*-means clustering which incorporates encryption of the numeric characters in a URL. *k*-means clustering is an optimal choice to apply when dataset instances are distinct or somewhat different in nature from each other. It is a fast and less complex algorithm in comparison with other clustering algorithms that provides optimal results. Moreover, *k*-means provides a set of the parameter that can be tuned to improve the clustering such as number of clusters (k), distance method and centroid position updation, etc. For experimentation, the hyperparameters are tuned for accurate clustering. In final setting, *n_clusters* is set to 2, init is set to 'k-means++', *max_iter* is set to 300, *n_init* is set to 10 and *random_state* is set to 0.

A specific URL is composed of alphanumeric format (i.e. $www.64Tech-new.com$), therefore we perform analysis on the structure of benign and malicious URLs. Some of the malicious URLs try to be depicted as the original URL based on domain or length of URL having a minor difference. Most of the URLs have more characters than a usual URL. Similarly, the usage of special characters in domains is some of the signs of malicious URL. To get the best performance of *k*-means clustering, we convert contiguous numeric characters into distinct numerical IDs. We encrypt each URL using the steps given in the algorithm below:

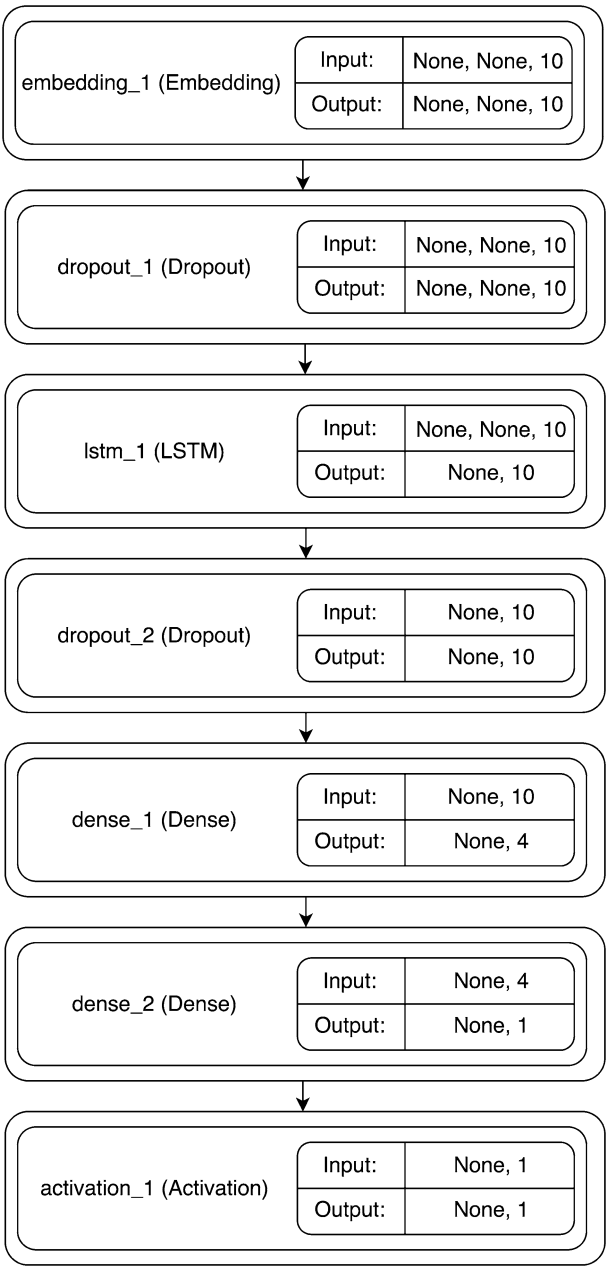


Fig. 4 The parameters used to train the LSTM Model

Algorithm 2 Algorithm for URL Encryption based k-means Clustering

Input: URL(s);
Output: Cluster of Malicious or Benign URL(s);

1: Calculate the length of the URL;
2: Count the number of digits in the URL;
3: Count the number of non-numeric characters in the URL;
4: Extract the domain name from URL;
5: Convert alphabets of domain name into digits while not changing digits already in URL;
6: Encrypted_URL ¡- join (domain name digit conversion + No. of Characters other than alphabets+ No. of Digits + Length of URL);
7: k-mean ¡- Encrypted_URL
8: **return:** Cluster of Malicious or Benign URL(s)

This unique URL encryption mechanism represents a URL and its characteristics based on an encoding which is used as a feature vector for URL clustering. After converting all the URLs into encrypted code, the dataset is given as input to the k-means clustering algorithm.

5 Analysis and Evaluation of Results

In this section, we discuss the results of our implementation and evaluate them by comparing with the conventional machine learning models. For experimentation, we perform supervised URL classification and unsupervised URL clustering. For supervised classification, we choose conventional algorithms: RF, MLP, NB, and LSTM, a deep learning method. For unsupervised learning, we perform URL encryption-based k-means clustering [47]. We also describe the evaluation measures that we have used to describe our results. We split our dataset into two parts for training and testing purposes. For training, we select 0.8 entries while for testing we keep 0.2 entries (training: 20%, testing: 80%). We feed the training data into our classification models for the prediction of URLs as malicious or benign.

5.1 Dataset Overview

We obtain a dataset of 2019,[4] which is created to tackle the problem of malicious URLs on the Internet. It is acquired from various sources such as PhishTank etc. It contains records of benign and malicious URLs that can be used for analysis or building classifiers. The total number of unique URLs in the dataset is 450176 out of which 77% are benign and 23% are malicious. The dataset contains four columns representing the index, URL, label, and result. We choose this dataset for our work because it has a variety of URLs from different sources. The size of different URLs vary from each other which means our model can detect the maliciousness of a URL with any variable length. The dataset also has a large number of entries of malicious and benign URLs

[4] https://www.kaggle.com/siddharthkumar25/malicious-and-benign-urls.

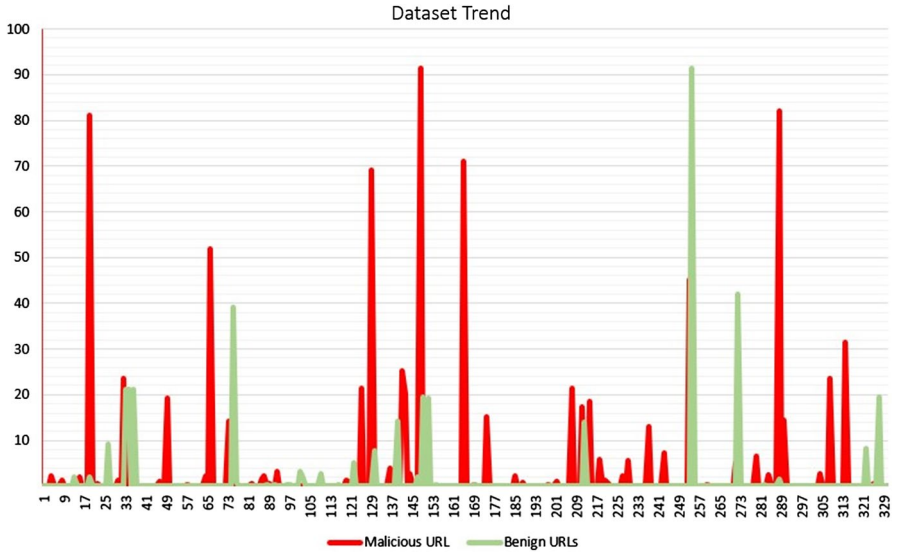


Fig. 5 The representation of dataset concerning the length of URLs

due to which the model has enough input for training and test. Hence, the above mentioned reasons make this dataset appropriate for our work.

To use the dataset for our work, we delete a few benign entries randomly to make the percentage of malicious and benign URLs equal i.e. 50% malicious and 50% benign to obtain better results. The final dataset consists of 207,081 entries with equal malicious and benign URLs. All the information from this dataset is used for training and testing purposes. For training purposes, 80% of the data is used while testing is done on the rest of the 20% data. Figure 5 depicts the trend of benign and malicious URLs in the dataset. This representation demonstrates the URLs concerning the URLs' length. The longer the URL, the more number of tokens it has usually resulting in helping the model to predict better accuracy.

5.1.1 Pre-processing of Data

Firstly, the dataset contains several NaN values so we remove a few rows. The dataset even after removing some rows is large enough for training and testing purposes. Secondly, the dataset contains benign entries in more numbers as compared to malicious entries. Therefore, we also remove some benign entries to make the ratio of benign and malicious entries equal in the dataset. This balances our data for classification that leads to the better catering of both the classes.

Table 2 Computing environment

Operating System	Windows 10 Professional 1909
CPU	Intel(R) Core(TM)i7-6700HQ
RAM	16GB
GPU	NVIDIA GeFroce 1060
CUDA Verion	9.0
Python Version	3.8

5.2 Evaluation Measures and Computing Environment

Practically all evaluation measures rely upon the nature of the dataset. For the most part, accuracy is taken as an essential measure yet it is for the situation when the dataset is balanced. In any case, when the dataset is imbalanced, it does not provide significant information about the performance of the model. In this paper, we consider the most valuable performance measures such as Accuracy, Precision, Recall, F-Score, and Confusion Matrix to analyze the performance of our classification model. Below, equations demonstrate the evaluation measures, and Table 2 presents the computing environment.

$$Accuracy = \frac{TP_{Malicious} + TN_{Benign}}{TP_{Malicious} + FN_{Malicious} + TN_{Benign} + FP_{Benign}} \tag{1}$$

$$Precision = \frac{TP_{Malicious}}{TP_{Malicious} + FP_{Malicious}} \tag{2}$$

$$Recall = \frac{TP_{Malicious}}{TP_{Malicious} + FN_{Benign}} \tag{3}$$

$$F - Score = 2 \times \frac{\frac{TP_{Malicious}}{TP_{Malicious}+FN_{Benign}} \times \frac{TP_{Malicious}}{TP_{Malicious}+FP_{Malicious}}}{\frac{TP_{Malicious}}{TP_{Malicious}+FN_{Benign}} + \frac{TP_{Malicious}}{TP_{Malicious}+FP_{Malicious}}} \tag{4}$$

Confusion matrix comprises of TP, TN, FP, and FN, where TP is a malicious URL that is correctly classified as malicious URL, TN is a benign URL that is correctly classified as benign URL, FP is a benign URL that is incorrectly classified as malicious URL, and FN is a malicious URL that is incorrectly classified as a benign URL.

Table 3 Parameters for training model

Parameters	Function/value
Optimizer	Adam
Loss function	Binary cross entropy
Dropout rate	0.2
Dense units	4
Epochs	10

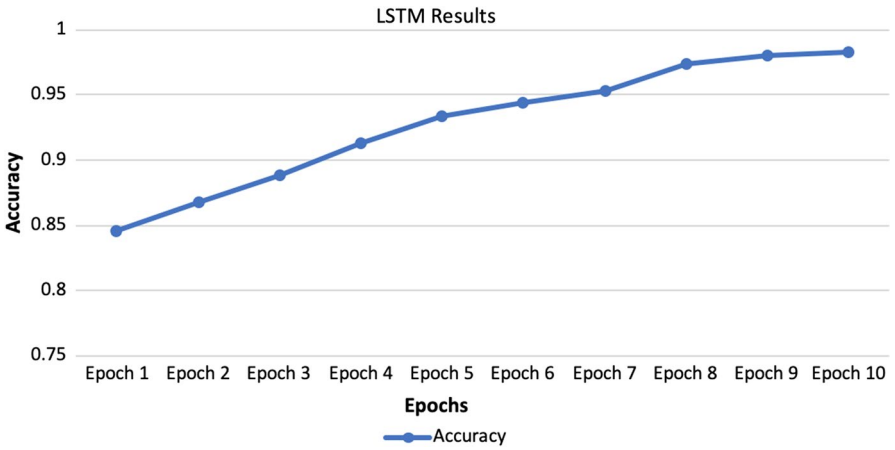


Fig. 6 Accuracy of LSTM model

5.3 Evaluation of Technique 1

In this section, we analyze the training parameters for the LSTM model and later discuss the results that we obtain via this model.

Parameter tuning is a critical part to obtain the best results of a machine learning model. The combination of parameters that produce the results is listed in Table 3 and demonstrated in Fig. 4.

We make use of an optimizer algorithm to reduce the error function in our model. It is important to do so because this is a vital step in the neural network's training process as it helps in the minimization of the loss function. The purpose of the loss function is to help the optimizer in determining whether the direction adopted by it is right in reaching the global minimum. In our approach, we use the optimizer named Adam because its usage is quite well in practice. Other than that, the convergence through this is very fast as compared to other optimization techniques. It performs very efficiently and makes sure that the problems encountered in other techniques are also rectified. The loss function we use for our classification problem is cross-entropy. This loss function is also preferred to be used mathematically. A score is calculated using this function which is then minimized to obtain a perfect value i.e. 0. We specify the cross-entropy in Keras by setting the loss to 'binary_crossentropy' when we compile the model. Next, we use a sigmoid activation function which has

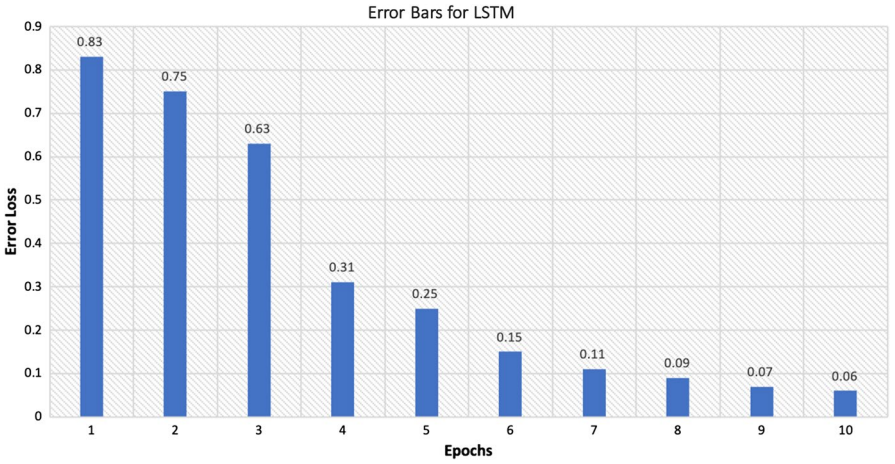


Fig. 7 Error bars for LSTM

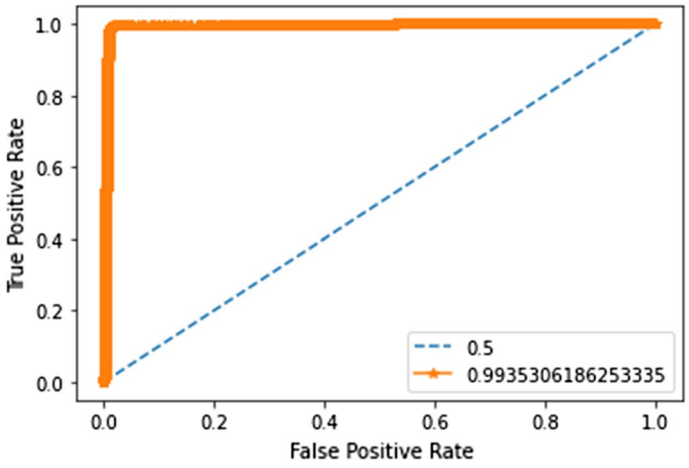


Fig. 8 ROC curve of LSTM

an S-Shape curve on the graph. The main reason behind using this function is that the values of the function lie between [0, 1]. Hence, we use it in our model to predict such values as the output. Next, we use a dense layer in our neural network. After that, we apply dropout to the input. Dropout is used to prevent over-fitting during training time. Lastly, we run a total of 10 epochs on LSTM. Each epoch has one forward pass and one backward pass of all the training examples in the dataset.

Figure 6 depicts the model accuracy achieved through LSTM. This model achieves the highest accuracy of 0.983 with a loss of 0.061 only which is depicted in Fig. 7. Figure illustrates the loss obtained during each epoch. We also obtain an f1-score of 0.983, while precision and recall are 0.990 and 0.977 respectively.

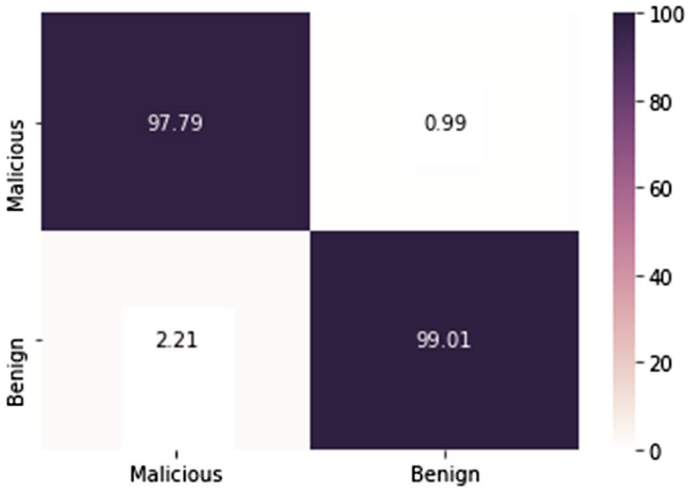


Fig. 9 Confusion matrix of LSTM

Table 4 Results of LSTM and *k*-means clustering

	Accuracy	F-1 Score	Precision	Recall
Long Short-Term Memory	0.983	0.983	0.990	0.977
k -Means clustering	0.997	0.968	0.970	0.957

Figure 8 demonstrates the ROC Curve for the model whereas Fig. 9 presents the confusion matrix of LSTM.

Similarly, Table 4 demonstrates the results of our proposed LSTM and *k*-means clustering. We use the metrics of accuracy, f-measure, precision, and recall to evaluate the results of these models.

5.4 Evaluation of Technique 2

Table 4 demonstrates the results of our proposed *k*-means using Embedding. We use the metrics of accuracy, f-measure, precision, and recall to evaluate the results of this model. k-Means clustering achieves the accuracy, F-Score, Precision, Recall of 0.997, 0.968,0.970,0.957 respectively, and provides promising results in comparison with LSTM for malicious URL detection. Figure 10 demonstrates a deep analysis of URLs through which we can cluster the sensitivity of a URL. The characterizing formula for a URL and *k*-means clustering shows that variance in data values is greater in malicious URLs as compare to benign URLs. This concludes that malicious URL format could be of multiple types like it could be resembling an original URL or it could be an absurd looking URL. This analysis suggests that the URL sensitivity can be categorized through its URL length.

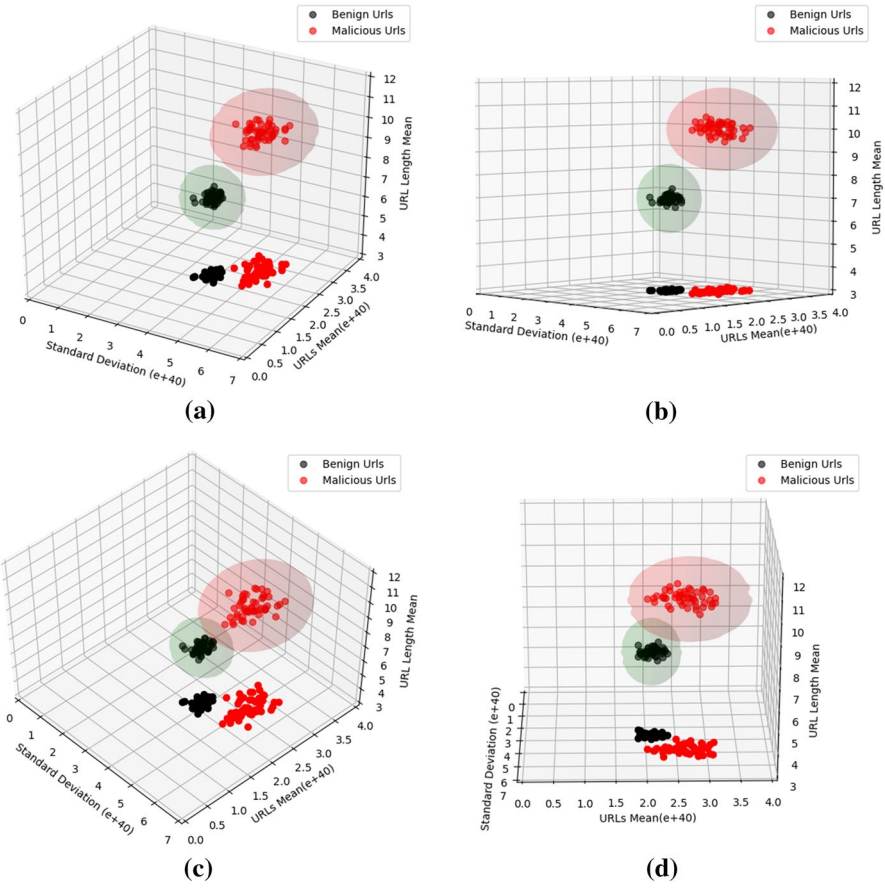


Fig. 10 This figure is a depiction of clusters in 3D space against standard deviation, URLs mean (characterization according to our formula) and URLs mean length, clustering a URL category with multi perspective

Figure 11 interprets and validates the consistency of *k*-mean within the cluster.

5.5 Evaluation of Conventional Classification Models

We train several other classification models to perform a comparison between the results obtained from Random Forest, Multilayer Perceptron, and Naïve Bayes with LSTM and k-Means clustering using encryption for URL classification. We use the sklearn classifiers to train these models on our dataset. Random Forest is trained by setting the parameters of *n_estimators* to 50, *max_depth* to 3, and *random_state* to 10. Similarly. Multilayer Perceptron is trained using the parameters of *hidden_layer_sizes* and *max_iter* by setting them to (2, 2, 2) and 10 respectively.

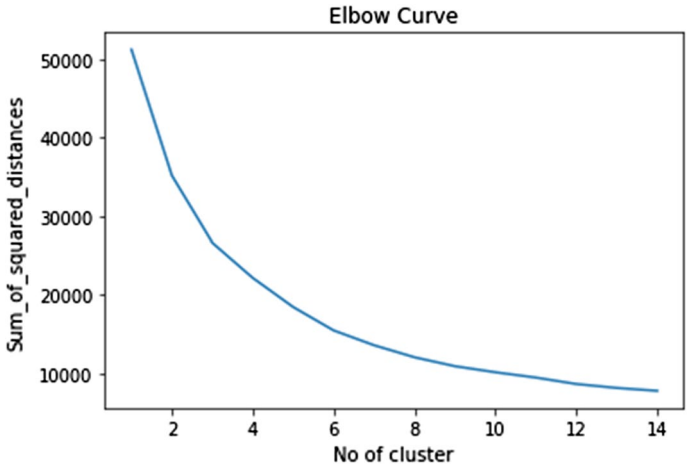
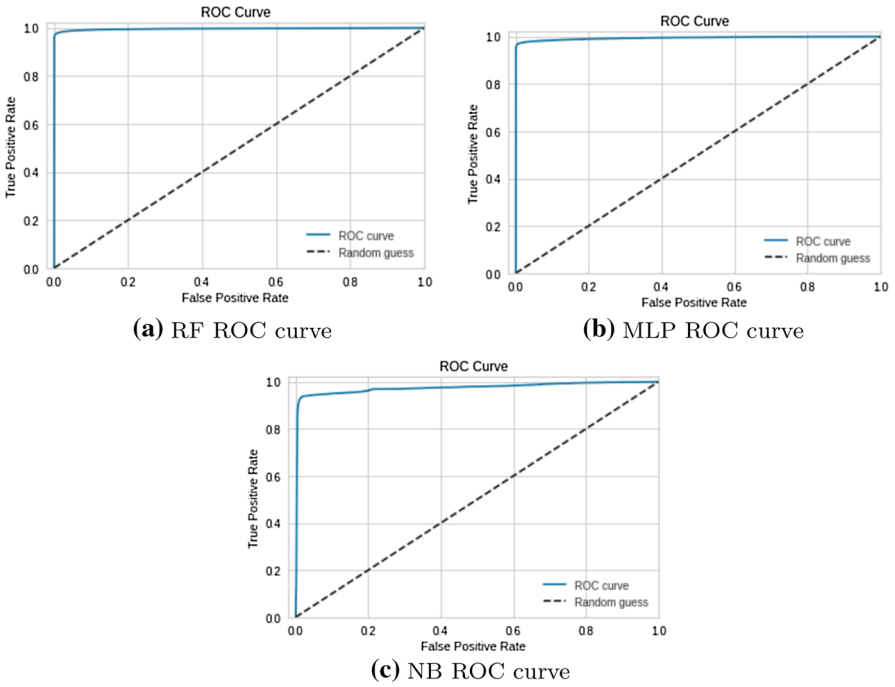


Fig. 11 Elbow curve for results validation with respect to various number of clusters



(a) RF ROC curve **(b)** MLP ROC curve

(c) NB ROC curve

Fig. 12 ROC curves for random forest, multilayer perceptron, and Naive Bayes

The classification models give an accuracy of 0.99, 0.98, and 0.97 in case of Random Forest, Multilayer Perceptron, and Naïve Bayes respectively. Figure 12 demonstrates the ROC curves for all three models.

Table 5 Results of all techniques

	Accuracy	F-measure	Precision	Recall
Random forest	0.993	0.991	0.991	0.984
Multilayer perceptron	0.982	0.984	0.994	0.979
Naive Bayes	0.956	0.955	0.992	0.922
LSTM	0.983	0.983	0.990	0.977
k -Means clustering	0.997	0.968	0.970	0.957

5.6 Comparative Analysis of Results

Table 5 presents the results in comparison with the conventional models. We use Random Forest, Multilayer Perceptron, and Naive Bayes for URL classification. Each of these models provide promising results on our dataset. We evaluate our models with different evaluation metrics which include accuracy, f-measure, precision and recall.

6 Conclusion and Future Work

In this article, we proposed two techniques which are LSTM and *k*-Means clustering to detect malicious URLs. Existing work on URL classification uses non-contiguous features such as bag-of-words and does not infer semantics from individual URL components. The limitations of those works include not maintaining a sequence of words, not interpreting useful meaning out of rare words in the URL, and not detecting unseen URLs correctly. To overcome these limitations, we worked with word embedding to generate a semantic vector model, and URL encryption to detect the real nature of the URLs. The novelty of our contributions include working with token-level embedding and URL encryption to feed the processed URLs into the model where they were classified as malicious or benign with an accuracy of 98.3% with a loss of 0.061 with LSTM and 99.7% with *k*-means clustering.

In the future, we can extend this work by developing an extension for real-time detection of URLs as malicious or benign. Such an extension would turn the color of the URL as red or green when the cursor is moved onto it depicting malicious or benign URL respectively. Furthermore, other deep learning models can be applied to the dataset to depict whether more accurate results can be achieved.

Compliance with ethical standards

Conflicts of interest The authors declare no conflict of interest.

References

1. Bakshy, E., Rosenn, I., Marlow, C., Adamic, L.: The role of social networks in information diffusion. In: Proceedings of the 21st international conference on World Wide Web, pp 519–528 (2012)

2. Basit, A., Zafar, M., Liu, X., Javed, A.R., Jalil, Z., Kifayat, K.: A comprehensive survey of ai-enabled phishing attacks detection techniques. Telecommunication Systems pp 1–16 (2020)
3. Asad, M., Asim, M., Javed, T., Beg, M.O., Mujtaba, H., Abbas, S.: Deepdetect: detection of distributed denial of service attacks using deep learning. Comput. J. **63**(7), 983–994 (2020)
4. Benevenuto, F., Magno, G., Rodrigues, T., Almeida, V.: Detecting spammers on twitter. In: Collaboration, electronic messaging, anti-abuse and spam conference (CEAS), vol 6, p 12 (2010)
5. Javed, A.R., Beg, M.O., Asim, M., Baker, T., Al-Bayatti, A.H.: Alphalogger: detecting motion-based side-channel attack using smartphone keystrokes. J. Ambient Intell. Human. Comput. pp 1–14 (2020)
6. Nair, M.C., Prema, S.: A distributed system for detecting phishing in twitter stream. Int. J. Eng. Sci. Innov. Technol. **3**(2), 151–158 (2014)
7. Leukfeldt, E.R., Kleemans, E.R., Stol, W.P.: Cybercriminal networks, social ties and online forums: social ties versus digital ties within phishing and malware networks. Br. J. Criminol. **57**(3), 704–722 (2017)
8. Ohta, S., Kurebayashi, R., Kobayashi, K.: Minimizing false positives of a decision tree classifier for intrusion detection on the internet. J. Netw. Syst. Manag. **16**(4), 399–419 (2008)
9. Jiang, J., Papavassiliou, S.: Detecting network attacks in the internet via statistical network traffic normality prediction. J. Netw. Syst. Manag. **12**(1), 51–72 (2004)
10. Joshi, A., Lloyd, L., Westin, P., Seethapathy, S.: Using lexical features for malicious url detection–a machine learning approach. arXiv preprint arXiv:191006277 (2019)
11. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: Proceedings of the 19th international conference on World wide web, pp 281–290 (2010)
12. Moshchuk, A., Bragin, T., Gribble, S.D., Levy, H.M.: A crawler-based study of spyware in the web. In: NDSS, vol 1, p 2 (2006)
13. Hofstede, R., Jonker, M., Sperotto, A., Pras, A.: Flow-based web application brute-force attack and compromise detection. J. Netw. Syst. Manag. **25**(4), 735–758 (2017)
14. Alshboul, Y., Nepali, R., Wang, Y.: Detecting malicious short urls on twitter. In: Conference: 21st Americas Conference on Information SystemsAt: Puerto Rico (2015)
15. Shafahi, M., Kempers, L., Afsarmanesh, H.: Phishing through social bots on twitter. In: 2016 IEEE International Conference on Big Data (Big Data), IEEE, pp 3703–3712 (2016)
16. Burnap, P., Javed, A., Rana, O.F., Awan, M.S.: Real-time classification of malicious urls on twitter using machine activity data. In: Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015, pp 970–977 (2015)
17. Lee, C.H.: Unsupervised and supervised learning to evaluate event relatedness based on content mining from social-media streams. Expert Syst. Appl. **39**(18), 13338–13356 (2012)
18. Imtiaz, S.I., ur Rehman, S., Javed, A.R., Jalil, Z., Liu, X., Alnumay, W.S.: Deepamd: Detection and identification of android malware using high-efficient deep artificial neural network. Future Generation Computer Systems (2020)
19. Nepali, R.K., Wang, Y.: You look suspicious!!: Leveraging visible attributes to classify malicious short urls on twitter. In: 2016 49th Hawaii International Conference on System Sciences (HICSS), IEEE, pp 2648–2655 (2016)
20. Kuyama, M., Kakizaki, Y., Sasaki, R.: Method for detecting a malicious domain by using whois and dns features. In: The third international conference on digital security and forensics (DigitalSec2016), vol 74 (2016)
21. Javed, A., Burnap, P., Rana, O.: Prediction of drive-by download attacks on twitter. Inf. Process. Manag. **56**(3), 1133–1145 (2019)
22. Jahani, H., Jalili, S.: Online tor privacy breach through website fingerprinting attack. J. Netw. Syst. Manag. **27**(2), 289–326 (2019)
23. Blum, A., Wardman, B., Solorio, T., Warner, G.: Lexical feature based phishing url detection using online learning. In: Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security, pp 54–60 (2010)
24. Cao, C., Caverlee, J.: Detecting spam urls in social media via behavioral analysis. In: European conference on information retrieval, Springer, pp 703–714 (2015)
25. Wang, D., Navathe, S.B., Liu, L., Irani, D., Tamersoy, A., Pu, C.: Click traffic analysis of short url spam on twitter. In: 9th IEEE International Conference on Collaborative Computing: Networking, pp. 250–259. Applications and Worksharing, IEEE (2013)

26. Verma, M., Sofat, S.: Techniques to detect spammers in twitter-a survey. Intl. J. Comput. Appl. **85**(10), (2014)
27. Selvaganapathy, S., Nivaashini, M., Natarajan, H.: Deep belief network based detection and categorization of malicious urls. Inf. Secur. J. **27**(3), 145–161 (2018)
28. Vinayakumar, R., Soman, K., Poornachandran, P.: Evaluating deep learning approaches to characterize and classify malicious url's. Journal of Intelligent & Fuzzy Systems **34**(3), 1333–1343 (2018)
29. Saxe, J., Berlin, K.: expose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. arXiv preprint arXiv:170208568 (2017)
30. Patgiri, R., Katari, H., Kumar, R., Sharma, D.: Empirical study on malicious url detection using machine learning. In: International Conference on Distributed Computing and Internet Technology, Springer, pp 380–388 (2019)
31. Begum, A., Badugu, S.: A study of malicious url detection using machine learning and heuristic approaches. In: Advances in Decision Sciences, pp. 587–597. Image Processing, Security and Computer Vision, Springer (2020)
32. Kulkarni, A.D., Brown, L.L., III.: Phishing websites detection using machine learning. Intl. J. Adv. Comput. Sci. Appl. **10**(7), (2019)
33. Zafar, S., Jangsher, S., Bouachir, O., Aloqaily, M., Othman, J.B.: Qos enhancement with deep learning-based interference prediction in mobile iot. Comput. Commun. **148**, 86–97 (2019)
34. Zafar, S., Jangsher, S., Aloqaily, M., Bouachir, O., Othman, J.B.: Resource allocation in moving small cell network using deep learning based interference determination. In: 2019 IEEE 30th Annual International Symposium on Personal, pp. 1–6. Indoor and Mobile Radio Communications (PIMRC), IEEE (2019)
35. Lee, S., Kim, J.: Warningbird: a near real-time detection system for suspicious urls in twitter stream. IEEE Trans. Depend. Secure Comput. **10**(3), 183–195 (2013)
36. Liew, S.W., Sani, N.F.M., Abdullah, M.T., Yaakob, R., Sharum, M.Y.: An effective security alert mechanism for real-time phishing tweet detection on twitter. Comput. Secur. **83**, 201–207 (2019)
37. Patil, D.R., Patil, J.B.: Feature-based malicious url and attack type detection using multi-class classification. ISeCure **10**(2), (2018)
38. Namasivayam, B.: Categorization of phishing detection features. PhD thesis, PhD thesis, Arizona State University (2017)
39. Hai, Q.T., Hwang, S.O.: Detection of malicious urls based on word vector representation and ngram. J. Intell. Fuzzy Syst. **35**(6), 5889–5900 (2018)
40. Yuan, H., Yang, Z., Chen, X., Li, Y., Liu, W.: Url2vec: Url modeling with character embeddings for fast and accurate phishing website detection. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), IEEE, pp 265–272 (2018)
41. Jang, B., Kim, I., Kim, J.W.: Word2vec convolutional neural networks for classification of news articles and tweets. PLoS ONE **14**(8), (2019)
42. Otoum, S., Kantarci, B., Mouftah, H.T.: On the feasibility of deep learning in sensor network intrusion detection. IEEE Netw. Lett. **1**(2), 68–71 (2019)
43. Aloqaily, M., Otoum, S., Al Ridhawi, I., Jararweh, Y.: An intrusion detection system for connected vehicles in smart cities. Ad Hoc Netw. **90**, 101842 (2019)
44. Javed, A.R., Usman, M., Rehman, S.U., Khan, M.U., Haghighi, M.S.: Anomaly detection in automated vehicles using multistage attention-based convolutional neural network. IEEE Trans. Intell. Transport. Syst. pp 1–10, https://doi.org/10.1109/TITS.2020.3025875 (2020)
45. Rehman Javed, A., Jalil, Z., Atif Moqurrab, S., Abbas, S., Liu, X.: Ensemble adaboost classifier for accurate and fast detection of botnet attacks in connected vehicles. Trans. Emerg. Telecommun. Technol. p e4088 (2020)
46. Le, H., Pham, Q., Sahoo, D., Hoi, S.C.: Urlnet: learning a url representation with deep learning for malicious url detection. arXiv preprint arXiv:180203162 (2018)
47. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: t-distributed stochastic neighbor embedding. J. Mach. Learn. Res. **12**, 2825–2830 (2011)

Sara Afzal is an Instructor at the Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan. She received her Bachelors degree in Computer Science from the same institution and is currently pursuing her Master's in Computer Science. Her research interests include but are not limited to Machine Learning and Network and Cyber Security.

Muhammad Asim is an Associate Professor at the Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan. Having attained a Ph.D. from Liverpool John Moores University, he researches in the fields of Cloud Computing, Computer Networks, Network Security, Internet of Things and Wireless Sensor Networks.

Abdul Rehman Javed is a lecturer at the Department of Cyber Security, Air University, Islamabad, Pakistan. He received his Master's degree in Computer Science from the National University of Computer and Emerging Sciences, Islamabad, Pakistan. His current research interests include but are not limited to mobile and ubiquitous computing, data analysis, knowledge discovery, data mining, natural language processing, smart homes, and their applications in human activity analysis, human motion analysis, and e-health.

Mirza Omer Beg is an Associate Professor at the Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan. His research interests lie at the confluence of Green Computation, Compiler Optimizations, Memory Hierarchy Optimizations, Constraint Programming and Graph Theoretic Algorithms. During the last few years he has worked on several compiler optimization problems targeted towards low-end, power constrained processors. These include formalizing instruction scheduling for clustered architectures, memory hierarchy optimizations and code generation.

Thar Baker is an Associate Professor at the Department of Computer Science, College of Computing and Informatics at University of Sharjah (UoS) in UAE. Before joining UoS, Thar was Reader in Cloud Engineering, Head of Applied Computing Research Group (ACRG), and International Mobility Head in the Department of Computer Science at Liverpool John Moores University (LJMU). He is Member of IEEE, BCS, ACM, Senior Fellow of UK Higher Education Academy (SFHEA), and a Member of Institute of Leadership and Management (iLM). Baker has published numerous refereed research papers and has been actively involved as a member of editorial board and review committee for a number of peer reviewed international journals and conferences.

Authors and Affiliations

Sara Afzal[1] · Muhammad Asim[1] · Abdul Rehman Javed[2] · Mirza Omer Beg[3] · Thar Baker[4]

Sara Afzal
sara.afzal@nu.edu.pk

Abdul Rehman Javed
abdulrehman.cs@au.edu.pk

Mirza Omer Beg
omer.beg@nu.edu.pk

Thar Baker
tshamsa@sharjah.ac.ae

[1] National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan

[2] Department of Cyber Security, Air University, Islamabad, Pakistan

[3] National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan

[4] Department of Computer Science, University of Sharjah, Sharjah 27272, UAE