Journal of Computer Science

Original Research Paper

Machine Learning and Deep Learning for Phishing Email Classification using One-Hot Encoding

[1*]Sikha Bagui, [2]Debarghya Nandi, [3]Subhash Bagui and [4]Robert Jamie White

[1]*Department of Computer Science, The University of West Florida, United States*
[2]*Division of Epidemiology and Biostatistics, School of Public Health, University of Illinois at Chicago, United States*
[3]*Department of Mathematics and Statistics, University of West Florida, United States*
[4]*AppRiver, Pensacola, FL, United States*

Article history
Received: 01-05-2021
Revised: 24-06-2021
Accepted: 28-06-2021

Corresponding Author:
Sikha Bagui
Department of Computer Science, The University of West Florida, United States
Email: bagui@uwf.edu

Abstract: Representation of text is a significant task in Natural Language Processing (NLP) and in recent years Deep Learning (DL) and Machine Learning (ML) have been widely used in various NLP tasks like topic classification, sentiment analysis and language translation. Until very recently, little work has been devoted to semantic analysis in phishing detection or phishing email detection. The novelty of this study is in using deep semantic analysis to capture inherent characteristics of the text body. One-hot encoding was used with DL and ML techniques to classify emails as phishing or non-phishing. A comparison of various parameters and hyperparameters was performed for DL. The results of various ML models, Naïve Bayes, SVM, Decision Tree, as well as DL models, Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM), were presented. The DL models performed better than the ML models in terms of accuracy, but the ML models performed better than the DL models in terms of computation time. CNN with Word Embedding performed the best in terms of accuracy (96.34%), demonstrating the effectiveness of semantic analysis in phishing email detection.

Keywords: One-Hot Encoding, Phishing Email Classification, Deep Learning, Machine Learning, Convolutional Neural Networks, Long Short Term Memory

Introduction

Phishing email attacks are intelligently crafted social engineering email attacks in which victims are conned by email to websites that impersonate legitimate sites. Phishing websites are generally well camouflaged. Victims of phishing email attacks perceive these sites to be associated with trusted companies such as Amazon or Google and hence are tricked into logging into such sites and sharing sensitive information. An estimated 269 billion emails are sent every day (Danny, 2020), with about one in every 2,000 being a phishing email, totaling 135 million phishing attacks attempted every day. Though the exact cost is very difficult to estimate, FBI has suggested that the impact of phishing attacks could be costing US businesses somewhere around $5 billion a year[1]. The Anti-Phishing Working Group (APWG)[2] reports that the most targeted sector is off course the payment sector (39.4%), webmail being the second highest (18.7%) and financial institutions being the third most targeted (14.2%) (APWG, 2021). Since phishing attacks affect millions of internet users (individuals as well as companies) and since APWG is reporting a continuous increase in unique phishing sites, it is becoming extremely important to seek ways to secure ourselves. Existing defense mechanisms need to be greatly improved (Behdad *et al*., 2012). Although many anti-phishing tools and techniques have been developed, phishing is still difficult to effectively defend, which puts the individual as well as organizations at risk (Fette *et al*., 2007). Behdad *et al*. (2012) points out that improving the defense mechanism is not enough and that systems should also be forward looking and intelligent to be able to identify fraudulent activities and prevent them from occurring.

Defense against phishing attacks is one of the hardest confronts faced (Thakur and Kaur, 2016). Though many email filters have been developed for spam emails, very few phishing email filters have been developed (Zhang *et al*., 2017). Of the phishing email filters that have been developed (Prakash *et al*., 2010; Cao *et al*., 2008; Ma *et al*., 2009a; Bergholz *et al*., 2010), ML filters (Bergholz *et al*., 2010) have achieved the best results so far.

Several traditional approaches for defense against phishing email attacks use various features that are static

in nature. Static features are not robust enough to handle new and emerging phishing patterns. Fraudsters are not static in their activities and are constantly changing the mode of operation to stay undetected. This motivates researchers into seeking effective techniques that can handle both known and emerging fraudulent patterns, hence the focus on machine learning algorithms (Akinyelu and Adewumi, 2014).

The novelty of this study is in applying deep semantic analysis with Deep Learning (DL) and Machine Learning (ML) to capture inherent characteristics of email text to classify emails as phishing or non-phishing. Until very recently, little work has been devoted to semantic analysis in phishing detection (Zhang *et al*., 2017), let alone phishing email detection. Representation of text is a significant task in Natural Language Processing (NLP) and in recent years DL has been widely used in various NLP tasks like topic classification, sentiment analysis and language translation (Zhang *et al*., 2017; LeCun *et al*., 2015). Word representations in NLP can be categorized into four classes: One-hot representations, distributional representations, clustering based word representations and distributed representations (Zhang *et al*., 2017; Lai *et al*., 2016). This study focuses on one-hot representations. Hence in this study, semantic analysis using one-hot encoding was used to preprocess the data, before using DL and ML techniques to classify emails. A comparison of the parameters and hyperparameters was performed for DL, Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM). The results of various ML algorithms, Naïve Bayes, SVM, Decision Tree and DL algorithms, CNN and LSTM, were presented in terms of accuracy and computation time. A comparison was also performed using CNN with Word Embedding, demonstrating the usefulness of semantic analysis using Word Embedding versus one-hot encoding.

The rest of this study is organized as follows. Section two presents the related works. Section 3 presents an overview of the methodology used. Sections four and five present the DL and ML algorithms respectively and sections six and seven present the results and conclusions respectively.

Related Works

Since phishing emails are becoming a significant threat, there are several recent works on phishing email detection. Yang *et al*. (2019) looked at phishing email detection based on 18 hybrid features including email-header structure, email-URL information and email-script function. SVM was used for classification and a classification accuracy of 95% was reached. Rawal *et al*. (2017) compared various classifiers, SVM, Random Forest, Logistic Regression, Naïve Bayes and Voted Perceptron, to classify emails as phishing or ham and received a maximum accuracy of 99.87%. Yasin and Abuhasan (2016) proposed an intelligent classification model of phishing email detection using a concept of phishing term weighting, which evaluates the weights of phishing emails in each email. Text stemming and WordNet ontology was also used. They achieve an accuracy of 99.1% using Random Forest and 98.4% using J48. Rastenis *et al*. (2021) presents a solution based on the email message body using text classification, classifying emails into spam and phishing emails. Fang *et al*. (2019) presented a recurrent CNN model with multilevel vectors and proposed a new phishing email detection model, Themis. Themis was used to model emails as email header, email body, character level and word level simultaneously. They reached a very high classification accuracy of 99.84%. Verma *et al*. (2020) looked at the classification of phishing emails using NLP.

The rest of this section focuses on works related to other techniques used in detecting phishing emails. These works can be divided into four broad categories: (i) Works that used black-list phishing detection methods; (ii) works that used heuristic phishing detection methods; (iii) works that used visual phishing detection methods; and (iv) works that used machine learning approaches.

Works that used Blacklist-based Phishing Detection Methods

The blacklist approach depends on building and maintaining a list of phishing websites. Though blacklist-based methods are easy to implement, they have the obvious deficiency of not being able to detect zero-hour phishing attacks, that is, attacks that have never happened before (Zhang *et al*., 2017).

Works that used Heuristic Phishing Detection Methods

Heuristic phishing detection identifies unknown web pages using features extracted from phishing attacks, but heuristic phishing rules are difficult to update because the rules are derived from statistical features or manual summaries (Zhang *et al*., 2017).

Yu *et al*. (2009) and Zhang *et al*. (2007) developed heuristic-based phishing detection systems achieving relatively low False Positive (FP) and False Negative (FN) rates. Prakash *et al*. (2010) used a combination of blacklist and heuristic methods to achieve low levels of FP and FN rates.

Works that used Visual Similarity Methods

Cao *et al*. (2008) and Bergholz *et al*. (2010) made use of features involving processing of images (for example, a company logo, copyright and other visual elements) to determine visual similarity, but this leads to higher run time and space needs.

Works that used Machine Learning Based Phishing Detection Methods

The key point of ML based phishing detection methods is to capture inherent patterns of phishing sites, hence the use of features. Basnet *et al*. (2008) used structural features in phishing emails, like the HTML format, IP-based address, age of domain name, number of domains, number of sub-domains, presence of Java Script, presence of form tags, number of links, URL based image source, matching domains and keywords, to classify emails using ML methods like SVM, neural networks, Self Organizing Maps (SOMs), K-means and ROC curves. Using k-means clustering, they achieved an accuracy of around 90%.

Chandrasekaran *et al*. (2006) extracted a total of 25 structural features for the classification of emails using SVM. They achieved 95% accuracy. Sarju and Thomas (2014) also used structural properties to detect spam emails, using Naïve Bayes, Adaboost and Random Forest for classification and comparison. Using a majority voting algorithm (using J48, SVM and IB1), they achieved a high accuracy of 99.8%. Bergholz *et al*. (2008) determined a feature set and used Random Forest and SVM to measure accuracy of detection. Ma *et al*. (2009b) extracted 43 features and used six different classifiers and found Random Forest outperforming the rest. Akinyelu and Adewumi (2014) also used a group of 15 features to classify emails using Random Forest. Fette *et al*. (2007) made use of ML techniques to receive a very low False Positive and False Negative rate.

Blum *et al*. (2010; Ma *et al*., 2009a; 2009b; Feroz and Mengel, 2014; 2015) proposed URL-based phishing detection methods, claiming accuracy of over 90%, but such measures turn out to be unstable since URLs can be manipulated easily. Xiang *et al*. (2011) proposed an improved URL-based method, but this method was overly dependent on third-party services.

Zhang *et al*. (2017) applied semantic analysis in phishing detection. This study focused on phishing websites rather than phishing emails and extracted various semantic features through word2vec to better describe the features of phishing sites. The authors achieved very low error rates and though semantic features analyzed with ML algorithms do a good job of analyzing phishing websites, little attention had been paid to semantic analysis before this study.

Eckhardt and Bagui (2021) used deep neural networks, Long Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) for phishing email classification. Though this study performed comparably to our method, this study did not take into account the processing time.

Though many of the above works have achieved high accuracy, none of them have analyzed the run time component, which we also do in this study. So, the main contributions of this study are:

- Determining if one-hot encoding is a useful pre-processing measure for phishing email classification
- Determining if Word Embedding is useful in phishing email classification (though this was only done with CNN)
- Using one-hot encoding, determining if ML or DL models are better in phishing-email classification
- Determining which parameters and hyperparameters would give better results in DL
- Determining which ML or DL models perform the best in terms of computation time using one hot-encoding

Methodology

Data

For this study, an email dataset, collected from App River, a company headquartered in Pensacola, Florida, USA, that offers secure cloud-based cybersecurity solutions, was used. This dataset contained 18,366 labelled emails, of which 3,416 were phishing emails and 14,950 were regular emails. The emails were collected across US industries - insurance, law, medical, hotel, school, banking and real estate marketing. Each of these emails contained a subject as well as body text and the number of users that this email was sent to. This study focuses on analyzing the text content of the emails and classifying them as to whether they were phishing emails or not. 70% of the data was used for training and the rest for testing.

Data Preprocessing: One-hot Encoding

ML algorithms cannot operate directly on textual data. Data has to be numeric. Hence, in this study, for data preprocessing, the email text was encoded as one-hot vectors. One-hot encoding uses a sparse vector in which one element is set to 1 and all other elements are set to 0 and is a commonly used technique to represent strings that have a finite set of values. Using one-hot encoding, high cardinality will lead to high dimensional feature vectors. But since one-hot encoding is simple, it is a widely-used encoding method (Davis, 2010; Cerda *et al*., 2018). One-hot encoding is effective for sentences or tweets that do not contain many repeated elements and is usually applied to models that have good smoothing properties. One-hot encoding is commonly used in neural networks, whose activation functions require input to be in the discrete range of [0,1] or [-1,1] (O'REILLY, 2021).

A one-hot vector is a $1 \times N$ matrix (vector) consisting of 0s in all cells of the vector with the exception of a single 1 in a cell used to uniquely identify a word. One hot encoding allows for more expressive representation of categorical data. A sample word range of [good, good, bad] would be represented in 3 such encodings [0, 0, 1], as shown in Table 1.

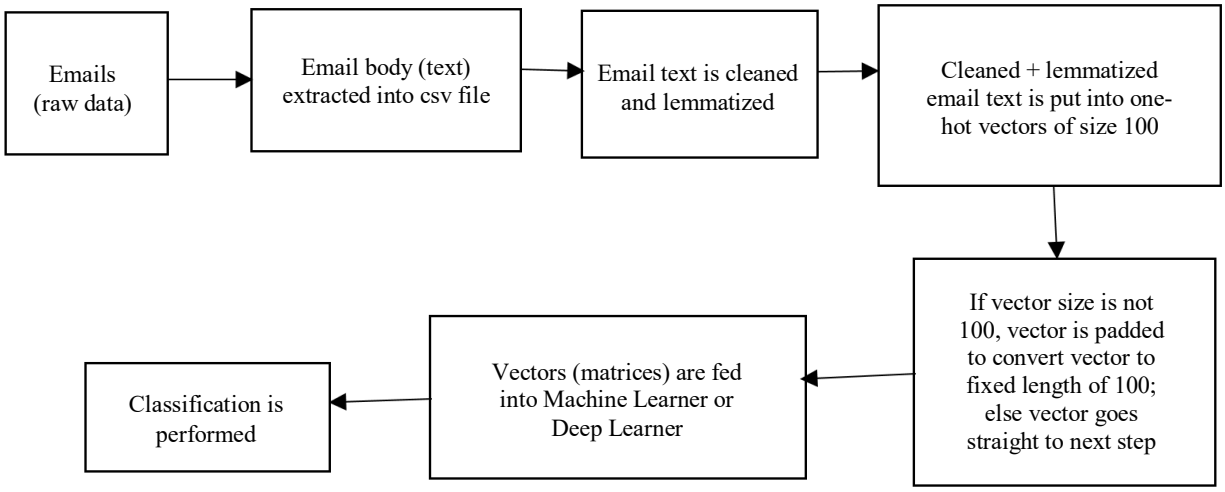

Fig. 1: Experimental flow

Table 1: One-hot encoding

1	0
1	0
0	1

In this study, the email text was cleaned, lemmatized and represented in the form of one-hot encoded vectors. Lemmatization is the process of grouping together the various forms of a word so that they can be analyzed as a single item (Allahyari *et al*., 2017).

Typically the length of the vector would depend on the total number of unique tokens. But, emails are of varying lengths and this is difficult to feed into the machine learner or neural network. Therefore, an arbitrary length of 100 was selected for this study and padding was done to convert all vectors to a fixed length. Sequences shorter than that length were padded with a pre-fixed value at the end and sequences longer than the threshold were truncated so that they would fit the desired length. These matrices were fed into the machine learner or neural network where these vectors were mapped to a low dimensional space. In the neural network, this low dimensional space is represented by a hidden layer where the number of words is lower than the vocabulary size.

Experimental Flow

Figure 1 presents the experimental flow for this study.

Deep Learning Algorithms

Two DL Algorithms, Long Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) were used. For these algorithms, experimentation was performed using the keras python library. Keras is built on the top of Tensor flow designed for fast testing of deep neural networks.

Long Short Term Memory

LSTM is an emerging and scalable model for learning sequential data (Palangi *et al*., 2016; Greff *et al*., 2017). LSTM has shown to be successful in sequence prediction (Adi *et al*., 2016), sequence labeling (Sak and Beaufays, 2014), syntactic structure (Linzen *et al*., 2016) and long range semantic dependencies (He *et al*., 2017). LSTM's have an edge over CNN and Recurrent Neural Networks (RNN) in many ways (Otte *et al*., 2014). RNNs are effective when working with short term dependencies, but fail to recognize context or chronologically widely spaced input events or long-term dependencies. RNNs cause the problem of the vanishing gradient. The vanishing gradient problem takes place when the weights in the matrix are so small that learning either becomes very slow or stops altogether. Conversely, if the weights in this matrix are large, it can lead to a situation where the gradient is so large that it can cause learning to diverge and this is called the exploding gradient problem. This mainly motivates the use of the LSTM model which introduces a new structure called the memory cell, protected from outer influences through surrounding gates.

The memory cell works with three dependencies: The previous cell state (that is, the information that was present in the memory after the previous time step), previous hidden state (the output of the previous cell) and input in the current step (new information that is being fed in). The memory cell performs the function of retaining values for periods of time. This is achieved by applying the no activation function on the memory cell unit, which in turn does not cause the gradient diminishing factor over time (Greff *et al*., 2017).

The memory cell is composed of four elements: An input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The self-recurrent connection of this memory cell has a weight of 1.0 that

prevents any outside interference, hence the state of the memory cell can remain constant from one time step to another (Srivastava, 2017). The gates use a logistic function to control the flow of data into and out of a cell. An input gate controls the extent to which new values flow into the cell, the forget gate controls the extent to which old values are retained in the cell and the output gate is the extent to which the value in the function is used to compute the output activation of the LSTM unit (Greff *et al*., 2017).

Architecture of our LSTM Model

Figure 2 presents the base LSTM model used in this study. The model consists of 2 LSTM layers with hidden nodes, followed by a dense layer of 1 node and a sigmoid activation layer.

Convolutional Neural Networks

CNN, a class of deep neural networks, has been widely used in classification in NLP. CNN consists of some combination of convolutional layers, pooling layers and a fully connected layer. The convolutional and pooling layers act as feature extractors and the fully connected layer acts as a classifier. One of the main advantages of CNN is that training can be performed without the need for engineered features. It is designed to effectively model multidimensional input data. It can learn intrinsic features in the raw dataset through the many layered structures which represent the different levels of abstraction of features. The layers are briefly explained below.

Convolutional Layer

In this study, the input is emails, represented as one-hot vectors and convolutions are used over the input layer. A convolutionary layer consists of multiple neurons or feature maps. Filters slide over rows of the matrix (words), performing convolutions on the one-hot vector and generating feature maps. Since all neurons in the feature map scan the same feature of the previous layer but from different locations, different feature maps detect different types of features (Choong and Lee, 2017).

Nonlinearity

The convolutional layer is typically followed by the non-linear activation function. The three most commonly used activation functions are sigmoid, tanh and Rectified Linear Unit (ReLU). ReLU(x) = max(0, input). ReLU is the most commonly used because of the non-vanishing gradient in the positive region and faster convergence compared to the other two activation functions (Akhtyamova *et al*., 2017). ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero (https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/).

Pooling

The pooling layers are typically applied after the convolutional layers. Pooling subsamples the input and this is commonly done by taking an average or maximum of small blocks of data, hence pooling reduces the spatial size of the representation. It also reduces the dimensionality of the output and provides a fixed size output matrix, which is typically required for classification. Pooling also controls overfitting.

Fully Connected Layer

Fully Connected means that every neuron in the previous layer is connected to every neuron in the next layer (https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/). After several convolutional and max-pooling layers, the outputs of these layers can be treated as new data representation. This can be flattened into a one-dimensional vector and used for classification. This layer is used for classification.

Kim (2014) used CNN to perform sentiment analysis and topic categorization. Johnson and Zhang (2014) applied convolutions directly to one-hot vectors, using a pre-training model that used word vectors like word2vec and GloVe. Johnson and Zhang (2015) extended Johnson and Zhang (2014)'s model with additional unsupervised region embeddings. Zhang and Wallace (2015) studied the effects of varying hyperparameters. Gao *et al*. (2019) and Shen *et al*. (2014) dealt with semantic representations of sentences.

Architecture of our CNN Model

Figure 3 presents the architecture of the CNN model used in this study. The input is one-hot vectors. The architecture consists of two convolutional_1D layers. The first convolutional layer has a filter size of 5 with a stride of one, with a total of 32 filters and the second convolutional layer has a filter size of 2 with a stride of one and the number of filters were increased to 64. Feature maps have been set to 4 in the first convolution layer and 8 in the second convolution layer and, the Relu activation function was used in the convolutional layers.

Followed by each convolution_1D layer is a maxpooling_1D layer. The max-pooling_ID layer performs down sampling using a spatial size of 2, with default stride. Max-pooling is the concept of taking windowed samples from the output of a convolutional layer and subsampling them to create a single output. The subsampling reduces the dimensionality of the input after each layer.

Followed by this is a fully connected dense layer with a single hidden node. The output of the previous layer is flattened to a [None by 1] shape. The Sigmoid activation function was used in the dense connected layer. This produces a total of 149 trainable parameters.

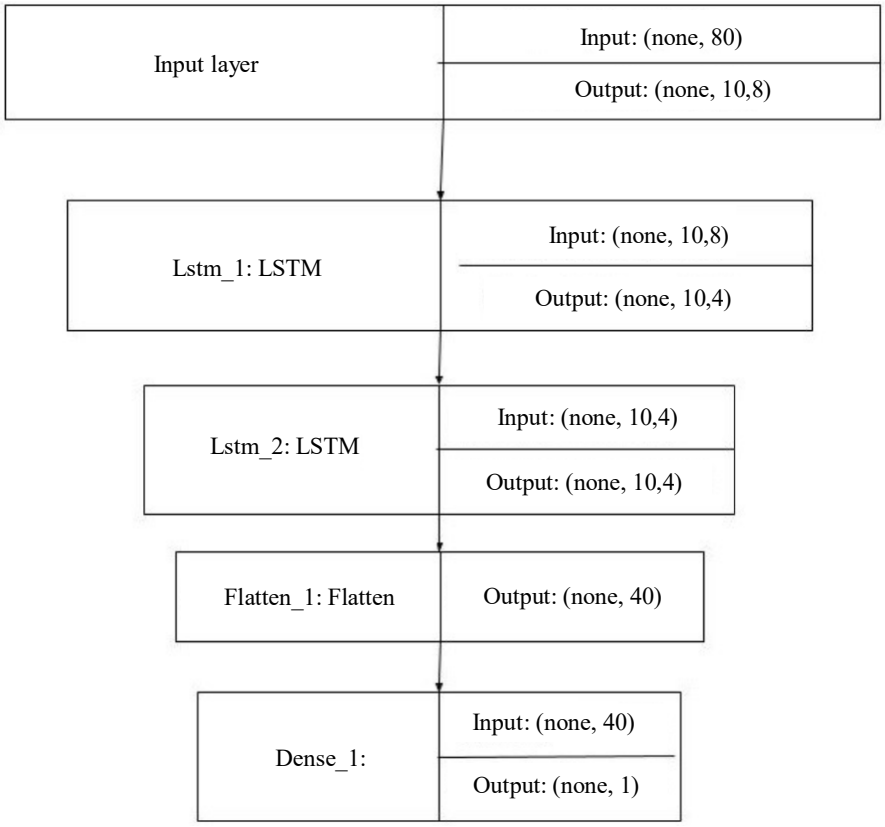

Fig. 2: Architecture of LSTM model

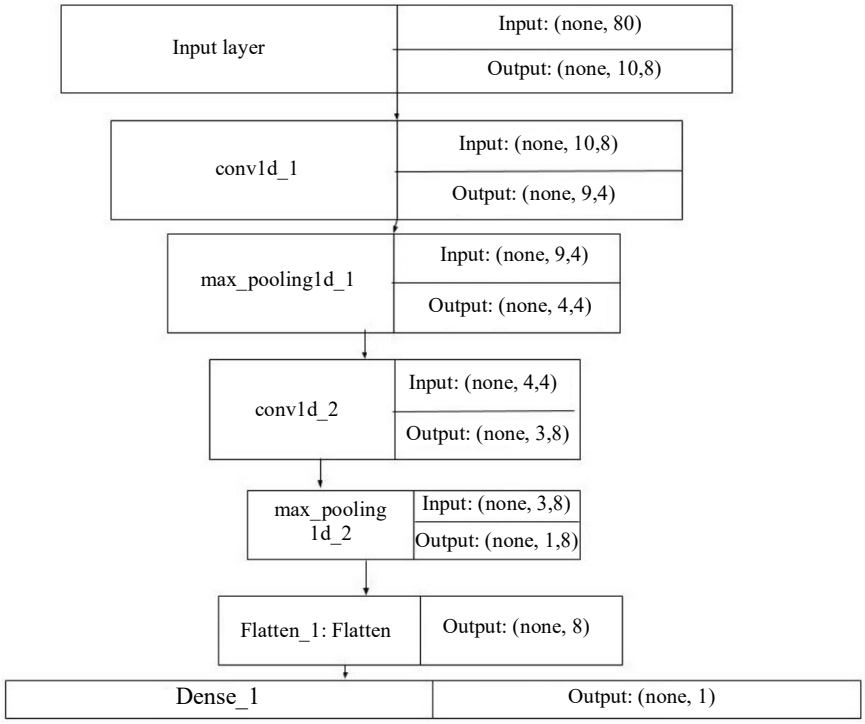

Fig. 3: Architecture of CNN model

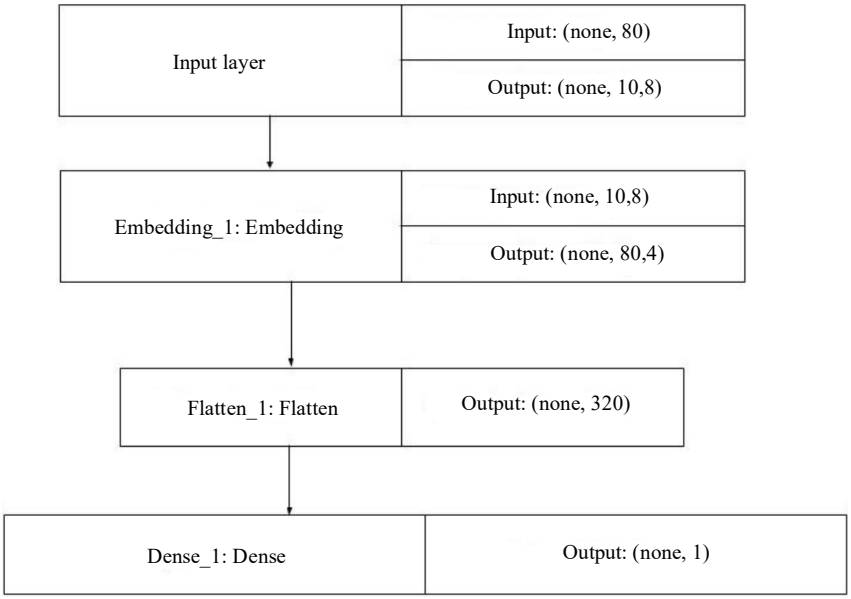

Fig. 4: Architecture of CNN with Word Embedding

Table 2: Word embedding semantic representation

Word	This	is	a	sample	CBOW	example
sample	0	0	1	0	0	0
sample	0	0	0	0	1	0

CNN with Word Embedding

CNN was also applied with Word Embedding. Word Embedding is a technique where individual words are represented as sparse vectors based on the context window and then mapped to a low dimensional vector space. Due to its inability to represent idiomatic phrases, vector representations of individual words are used to make the model more expressive. Continuous Bag of Words (CBOW) is a popular form of Word Embedding. Using Word Embedding, words with similar meaning will have similar representation. To understand the semantic representation of words in the email text, that is, the representation of the context of the word, CBOW, was used. A context may be a single word or a group of words based on the context window. Each word, represented in the form of one-hot encoding, can have multiple data points as context, as shown in Table 2. In Table 2, "Sample" in the text "This is a sample CBOW example" would have two contexts with a 1-word context window. Another example demonstrating semantic representation would be, in the sentence "Taj Mahal in India is one of the seven wonders of the world," "Taj Mahal" has a high probability of having a semantic similarity with 'India' rather than any other country, because they usually appear in the same context. This allows it to distinguish words with different meanings, for example 'crane' (a bird; an instrument used to lift heavy objects; to crane your neck etc.)

Architecture of our CNN with Word Embedding Model

The CNN architecture with Word Embedding used in this study is represented in Fig. 4. This consists of an input layer, an embedding layer with hidden nodes and finally a dense layer with a single hidden node. A sigmoid activation function is used in the final layer.

Machine Learning Models

The machine learning algorithms, Naïve Bayes, SVM and Decision Tree, that have been previously used in text classification, were used for comparison with the DL methods. These algorithms were implemented using (https://www.cs.waikato.ac.nz/ml/weka/).

Naive Bayes

The Naïve Bayes classifier assumes class independence. That is, an attribute value on a given class is independent of the values of the other attributes (Han *et al*., 2011). In a sense, this simplifies the computations involved and being relatively robust, easy to implement, fast and accurate, naïve Bayes classifiers have been used in many different fields including text classification (Zhang and Gao, 2011; Gupta and Lehal, 2009) and sentiment analysis (Vadivukarassi *et al*., 2017). Vadivukarassi *et al*. (2017) used an auxiliary feature to reclassify the text space for classification.

Support Vector Machines

Support Vector Machines (SVM) is a supervised learning model widely used for classification, known to be efficient in high dimensional spaces. Even with a small training set, SVM can provide high performance. Each sample is viewed as a data point in space. It constructs a hyperplane or a set of hyperplanes, which are used to categorize data into multiple classes. The larger the margin of separation, the lesser is the generalization error of the model. In some cases, the points are aligned in such a manner that it is physically impossible to separate them in space. In such cases, the points are mapped to a much higher dimension, thus increasing the scope for separation.

With a training data set of the form (x_1, y_1), … , (x_n, y_n), each point is represented as a p-dimensional real vector. The "maximum margin hyperplane" divides the group of points x_i into different class values so that the distance between the points in different segments and the hyperplane is maximized as much as possible.

A hyperplane can be defined by the formula: $\omega x - b = 0$, where ω is the normal vector to the hyperplane.

The parameter (b/ω) is the offset of the hyperplane from the origin.

The primary aim is to reduce this function:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-y_i\left(\vec{\omega}\cdot\vec{x}_i-b\right)\right)\right]+\lambda\|\vec{\omega}\|^2,$$

Many works have used SVM for text categorization (Joachims, 1998; Shin and Paek, 2018; Pilászy, 2005; Fatima and Srinivasu, 2017; Tong and Koller, 2001). SVM was used in this project because of the following SVM features (Joachims, 1998), which are applicable to our work:

(i) SVM can handle high dimensional input space
(ii) SVM does not need to remove irrelevant features through feature selection - it can use all the features, since in this case most of the features will contain some information
(iii) SVM works well with sparse vectors
(iv) Text categorization is linearly separable

Decision Tree

Decision Tree is a popular ML tool that uses a tree-like structure to represent all the chance events and their possible outcomes in different conditions. The advantages of decision tree are fast speed, relatively high accuracy and easily understood classification model. But, decision trees have a problem with high dimensional data, leading to low algorithm efficiency. Decision tree calculations are based on Information Gain and the attribute with the highest information gain is at the root of the tree, the attribute with the next highest information gain is at the next level of the tree and so forth. The leaves of the tree are the final decisions of the classifier (Han *et al*., 2011).

Results and Discussion

In this section, first we present an analysis of the hyper parameters. Then the results of the LSTM, CNN and CNN with Word Embedding are presented. Each of the results presented are an average of ten runs.

Analysis of Hyper Parameters

Proper estimation of hyperparameters is the key to optimizing any DL model. As pointed out in (Sarju and Thomas, 2014), selection of hyperparameters like context window, filter size, number of feature maps and pooling strategies, are completely dependent on the nature of the dataset. Plenty of trial runs were performed to come up with optimal values suited for the phishing dataset. The optimal values were:

For LSTM

The number of hidden nodes mattered the most. To determine the best accuracy, the number of hidden nodes was changed with each run.

For CNN

The optimum filter size (2), pool size (2), strides (1) and an activation function of 'relu' has been used for the convolution layers. Accuracy gradually improved with the increase in the number of feature maps.

For CNN with Word Embedding

The number of embedding nodes played a primary role in this accuracy.

All DL models were compiled using the *Adam* Optimizer and a loss function of *Binary Cross Entropy*. Though Relu has been the most effective activation function because it does not suffer from the diminishing gradient problem, sigmoid was used for the end-Dense layer. It performed well for the binary data.

Long Short Term Memory Results

To determine the performance of the LSTM model, the number of hidden nodes were varied and accuracy was determined with and without dropouts. Dropout refers to removing a random selection of units (hidden as well as visible) in a network layer for a single gradient step. The more units dropped out, the stronger the regularization. Dropouts prevent overfitting. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For input units, however, the optimal probability of retention is usually closer to 1 rather than to 0.5 (Srivastava *et al*., 2014). The LSTM experimental results are presented in Table 3.

Table 3: LSTM results

Hidden_nodes	Parameters	Computation_time	Accuracy (w/o dropout)	Accuracy (with dropout)
4	393	202.3	88.41	89.18
8	1169	204.9	91.8	89.05
16	3873	193.7	93.79	94.08
32	13889	244.5	94.77	94.68
64	52353	328.6	95.25	95.54
128	203009	696.1	95.7	95.91

Accuracy is defined as: *(True Positives + True Negatives)/Total Number*

The highest accuracy was achieved without dropout at 95.7% and with dropout at 95.91%, at 128 hidden nodes. The lowest computational time, however, was at 16 hidden nodes, at 193.7 ms. LSTM generally performed better with dropouts.

Convolutional Neural Network Results

Proper estimation of hyperparameters is the key to optimizing a CNN model. Selection of hyperparameters like context window, filter size, number of feature maps and pooling strategies are dependent on the nature of the dataset. Tables 4-7 present experimentation results with various context window sizes, filter sizes, embedding window sizes and pooling sizes, using the baseline model.

Baseline Model used

One-hot-size = 100
Filter Size = 5
Pool Size = 2
Activation Function = Relu
Feature Maps = 32

Base Model Structure used

Embedding Layer -- Conv1D -- MaxPool1D -- Conv1D -- MaxPool1D -- Flatten -- Dense (1)

A stride of one was used. Stride is how much the filter is shifted at each step. If the stride is 1, the filters overlap. A larger stride leads to fewer applications of the file and a smaller input size. From Tables 4-7, the following can be observed:

- The filter size of 7 produces the best accuracy
- Increasing the size of context window improves accuracy, but increasing it beyond a certain limit tends to increase computation time
- Increasing the pool size up to a size of 4 increases the accuracy, after which the accuracy drops.

Hence, for the final runs, varying the feature map sizes, a filter size of 7, context window of 100, embedding window of 80 and pooling size of 4 was used. The computation time and accuracy (with and without dropouts) was determined for various feature map sizes, as presented in Table 8.

The lowest computation time of 46.9 ms was with feature map size of 4 with 149 parameters. The highest accuracy, both without and with dropouts, at 94.98% and 95.97% respectively, was at feature map size of 64 with 17,729 parameters. Using CNN with one-hot encoding, the accuracy did not generally improve with dropouts.

Convolutional Neural Networks with Word Embedding

Table 9 presents the accuracy of varying the embedding nodes, with and without dropouts when CNN was performed with Word Embedding.

The lowest computation time of 45.7 ms was achieved with 4 Embedding Nodes and 721 parameters. The highest accuracy without dropout was at 96.1% with 32 embedding nodes and 5,761 parameters and at 96.34% with 64 nodes and 11,521 parameters. For CNN with Word Embedding, the accuracy generally improved slightly with dropouts.

Comparison of the Machine Learners as well as Deep Learners

Figure 5 graphically presents a comparison of the accuracy of the various models used for classification, Naïve Bayes, SVM, Decision Tree, LSTM, CNN and CNN with Word Embedding. One-hot encoding was used with the Naïve Bayes, SVM, Decision Tree, LSTM and CNN models. CNN with one-hot encoding achieved the highest classification accuracy of all the models using one-hot encoding, with an accuracy of 95.97%. However, CNN with Word Embedding performed better than CNN with one-hot encoding, with the highest overall accuracy of 96.34%.

Figure 6 presents the computation time taken for the various models. One-hot encoding was used to run Naïve Bayes, SVM, Decision Tree, LSTM and CNN. Naïve Bayes had the lowest computation time at 17.7 ms and LSTM had the highest computation time at 696.1 ms and, in this case CNN with Word Embedding did not perform particularly better.

Table 4: Accuracy by context window

Context window	Accuracy (%)
50	95.739
70	96.27
90	96.43
110	96.048
150	96.44

Table 5: Accuracy by filter size

Filter size	Accuracy (%)
3	96.376
5	96.522
7	96.592
9	95.81
11	96.103

Table 6: Accuracy by embedding layer nodes

Embedding window	Accuracy (%)
10	95.95
20	96.394
30	96.667
40	96.7
80	97.3
100	96.64

Table 7: Accuracy by pooling size

Pooling size	Accuracy (%)
2	96.44
3	96.59
4	96.9
5	96.3

Table 8: Computation time and accuracy by feature map size with and without dropouts

Feature maps	Parameters	Computation_time	Accuracy (w/o dropout)	Accuracy (with dropout)
4	149	46.9	85.25	81.6
8	425	52.1	88.65	86.74
16	1361	52.8	92.02	91.27
32	4769	59.4	92.53	93.718
64	17729	75.4	94.98	95.97

Table 9: Computation time and accuracy by embedding nodes with and without dropouts

Embedding nodes	Parameters	Computation_time	Accuracy (w/o dropout)	Accuracy (with dropout)
4	721	45.7	94.75	95.41
8	1441	49.8	95.5	95.68
16	2881	53.5	95.99	96.048
32	5761	73.7	96.1	96.23
64	11521	102.9	96.01	96.34

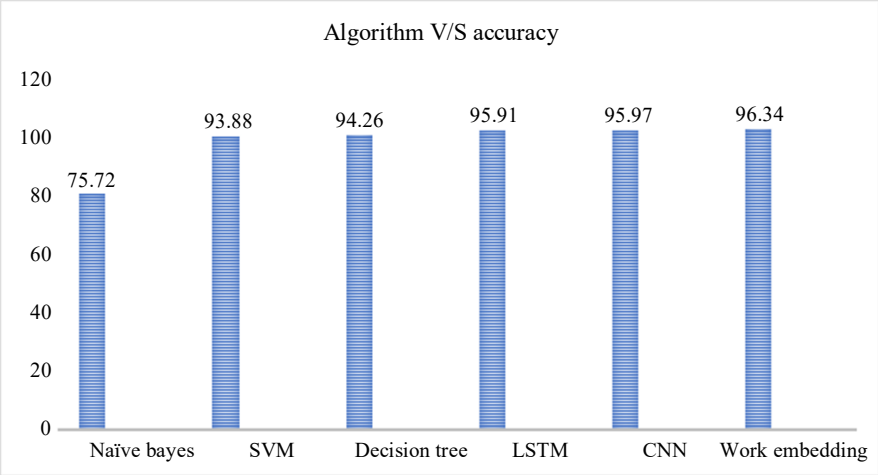

Fig. 5: Accuracy for the various models

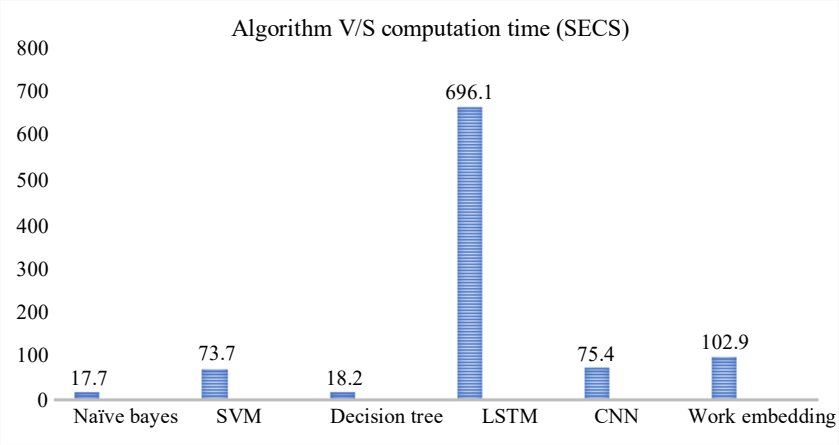

Fig. 6: Computation time for the various models

Conclusion

A comparison of ML and DL models was performed using one-hot encoding for preprocessing the data. From an analysis of the hyperparameters, for the CNN model, the best model was obtained for a filter size of 7, context window of 100, embedding window of 80 and pooling size of 4. For LSTM, hidden nodes seemed to matter the most, hence the hidden nodes were varied to determine the best accuracy with and without dropouts. LSTM generally performed better with dropouts. But, for CNN with one-hot encoding, the accuracy did not improve with dropouts, while for CNN with Word Embedding, the accuracy generally improved slightly with dropouts. Using one-hot encoding, CNN performed the best of all the models, with an accuracy of 95.97%. But CNN with Word Embedding performed better than CNN with one-hot encoding (96.34%). This demonstrates the power of semantic analysis in classifying phishing emails. As compared to previous works, these results are on the higher side, but an exact comparison may not be warranted since different techniques were used in each work. Moreover, none of the previous works analyzed the computation time, which was analyzed in this study in addition to accuracy.

In terms of computation time, Naïve Bayes (with one-hot encoding) performed the best at 17.1 ms. LSTM (with one hot encoding) performed very poorly in terms of computation time, though it's accuracy was pretty close to CNN with one-hot encoding. Finally, the DL models performed better than the ML models in terms of accuracy, but the ML models performed better than the DL models in terms of computation time.

Acknowledgement

This work has been partially supported by the Askew Institute of the University of West Florida.

Author's Contributions

Sikha Bagui: Formulated the project and the paper.

Debarghya Nandi: Also helped in the formulation of the project and in the programming part of this project.

Subhash Bagui: Helped in analysis and fine tuning of the algorithms.

Robert Jamie White: Also helped in the formulation of the project.

Ethics

No part of this manuscript is being considered for publication in whole or in part elsewhere. All authors have read and approved the manuscript.

References

[5]https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[6]Weka 3: Machine Learning Software in Java. https://www.cs.waikato.ac.nz/ml/weka/

Adi, Y., Kermany, E., Belinkov, Y., Lavi, O., & Goldberg, Y. (2016). Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. arXiv preprint arXiv:1608.04207. https://arxiv.org/abs/1608.04207

Akhtyamova, L., Alexandrov, M., & Cardiff, J. (2017, August). Adverse drug extraction in twitter data using convolutional neural network. In 2017 28th International Workshop on Database and Expert Systems Applications (DEXA) (pp. 88-92). IEEE. https://doi.org/10.1109/DEXA.2017.34

Akinyelu, A. A., & Adewumi, A. O. (2014). Classification of phishing email using random forest machine learning technique. Journal of Applied Mathematics, 2014.

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. arXiv preprint arXiv:1707.02919. https://arxiv.org/abs/1707.02919

APWG. (2021). Phishing activity trends reports https://apwg.org/

Basnet, R., Mukkamala, S., & Sung, A. H. (2008). Detection of phishing attacks: A machine learning approach. In Soft computing applications in industry (pp. 373-383). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77465-5_19

Behdad, M., Barone, L., Bennamoun, M., & French, T. (2012). Nature-inspired techniques in the context of fraud detection. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), 42(6), 1273-1290.

Bergholz, A., Chang, J. H., Paass, G., Reichartz, F., & Strobel, S. (2008, August). Improved Phishing Detection using Model-Based Features. In CEAS.

Bergholz, A., De Beer, J., Glahn, S., Moens, M. F., Paaß, G., & Strobel, S. (2010). New filtering approaches for phishing email. Journal of computer security, 18(1), 7-35. https://doi.org/10.3233/JCS-2010-0371

Blum, A., Wardman, B., Solorio, T., & Warner, G. (2010, October). Lexical feature based phishing URL detection using online learning. In Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security (pp. 54-60). https://doi.org/10.1145/1866423.1866434

Cao, Y., Han, W., & Le, Y. (2008, October). Anti-phishing based on automated individual white-list. In Proceedings of the 4th ACM workshop on Digital identity management (pp. 51-60). https://doi.org/10.1145/1456424.1456434

Cerda, P., Varoquaux, G., & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. Machine Learning, 107(8), 1477-1494. https://doi.org/10.1007/s10994-018-5724-2

Chandrasekaran, M., Narayanan, K., & Upadhyaya, S. (2006, June). Phishing email detection based on structural properties. In NYS cyber security conference (Vol. 3). https://www.albany.edu/wwwres/conf/iasymposium/proceedings/2006/chandrasekaran.pdf

Choong, A. C. H., & Lee, N. K. (2017, November). Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method. In 2017 International Conference on Computer and Drone Applications (IConDA) (pp. 60-65). IEEE. https://doi.org/10.1109/ICONDA.2017.8270400

Danny, P. (2020). What is Phishing? Everything you need to know to protect yourself from scam emails and more. https://www.zdnet.com/article/what-is-phishing-how-to-protect-yourself-from-scam-emails-and-more

Davis, M. J. (2010). Contrast coding in multiple regression analysis: Strengths, weaknesses and utility of popular coding structures. Journal of Data Science, 8(1), 61-73. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.4179&rep=rep1&type=pdf

Eckhardt, R. & Bagui, S. (2021). Convolutional Neural Networks and Long Short Term Memory for Phishing Email Classification. International Journal of Computer Science and Information Security, 19(5), 27-35. https://doi.org/10.5281/zenodo.4898110

Fang, Y., Zhang, C., Huang, C., Liu, L., & Yang, Y. (2019). Phishing email detection using improved RCNN model with multilevel v https://doi.org/10.1109/ACCESS.2019.2913705

Fatima, S., & Srinivasu, B. (2017). Text Document categorization using support vector machine. International Research Journal of Engineering and Technology (IRJET), 4(2), 141-147.

Feroz, M. N., & Mengel, S. (2014, October). Examination of data, rule generation and detection of phishing URLs using online logistic regression. In 2014 IEEE International Conference on Big Data (Big Data) (pp. 241-250). IEEE. https://doi.org/10.1109/BigData.2014.7004239

Feroz, M. N., & Mengel, S. (2015, June). Phishing URL detection using URL ranking. In 2015 ieee international congress on big data (pp. 635-638). IEEE. https://doi.org/10.1109/BigDataCongress.2015.97

Fette, I., Sadeh, N., & Tomasic, A. (2007, May). Learning to detect phishing emails. In Proceedings of the 16th international conference on World Wide Web (pp. 649-656). https://doi.org/10.1145/1242572.1242660

Gao, J., Pantel, P., Gamon, M., He, X., & Deng, L. (2019). Modeling interestingness with deep neural networks. https://doi.org/10.3115/v1/D14-1002

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmid Huber, J. (2017). LSTM: A Search Space Oddyssey. IEEE Transactions on Neural Networks and Learning Systems, 28, pp. 222-2232. https://doi.org/10.1109/TNNLS.2016.2582924

Gupta, V., & Lehal, G. S. (2009). A survey of text mining techniques and applications. Journal of emerging technologies in web intelligence, 1(1), 60-76. https://doi.org/10.4304/jetwi.1.1.60-76

Han, J., Pei, J., & Kamber, M. (2011). Data mining: concepts and techniques. Elsevier. ISBN-10: 0123814804.

He, L., Lee, K., Lewis, M., & Zettlemoyer, L. (2017, July). Deep semantic role labeling: What works and what's next. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 473-483). https://doi.org/10.18653/v1/P17-1044

Joachims, T. (1998, April). Text categorization with support vector machines: Learning with many relevant features. In European conference on machine learning (pp. 137-142). Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0026683

Johnson, R., & Zhang, T. (2014). Effective use of word order for text categorization with convolutional neural networks. arXiv preprint arXiv:1412.1058. https://doi.org/10.3115/v1/N15-1011

Johnson, R., & Zhang, T. (2015). Semi-supervised convolutional neural networks for text categorization via region embedding. Advances in neural information processing systems, 28, 919. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4831869/

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Process (EMNLP 2014), pp. 1746-1751. https://doi.org/10.3115/v1/D14-1181

Lai, S., Liu, K., He, S., & Zhao, J. (2016). How to generate a good word embedding. IEEE Intelligent Systems, 31(6), 5-14. https://doi.org/10.1109/MIS.2016.45

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

Linzen, T., Dupoux, E., & Goldberg, Y. (2016). Assessing the ability of LSTMs to learn syntax-sensitive dependencies. Transactions of the Association for Computational Linguistics, 4, 521-535. https://doi.org/10.1162/tacl_a_00115

Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009a, June). Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1245-1254). https://dl.acm.org/doi/abs/10.1145/1557019.1557153

Ma, L., Ofoghi, B., Watters, P., & Brown, S. (2009b, July). Detecting phishing emails using hybrid features. In 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing (pp. 493-497). IEEE. https://doi.org/10.1109/UIC-ATC.2009.103

O'REILLY. (2021). Applied Text Analysis with Python. Chapter 4. Text Vectorization and Transformation Pipelines. One-Hot Encoding. https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html#atap_ch04_one_hot_encoding

Otte, S., Liwicki, M., & Krechel, D. (2014, July). Investigating long short-term memory networks for various pattern recognition problems. In International Workshop on Machine Learning and Data Mining in Pattern Recognition (pp. 484-497). Springer, Cham. https://doi.org/10.1007/978-3-319-08979-9_37

Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., ... & Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. IEEE/ACM Transactions on Audio, Speech and Language Processing, 24(4), 694-707. https://doi.org/10.1109/TASLP.2016.2520371

Pilászy, I. (2005, November). Text categorization and support vector machines. In Proceedings of the 6th international symposium of Hungarian researchers on computational intelligence. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.8812&rep=rep1&type=pdf

Prakash, P., Kumar, M., Kompella, R. R., & Gupta, M. (2010, March). Phishnet: predictive blacklisting to detect phishing attacks. In 2010 Proceedings IEEE INFOCOM (pp. 1-5). IEEE. https://doi.org/10.1109/INFCOM.2010.5462216

Rastenis, J., Ramanauskaitė, S., Suzdalev, I., Tunaitytė, K., Janulevičius, J., & Čenys, A. (2021). Multi-Language Spam/Phishing Classification by Email Body Text: Toward Automated Security Incident Investigation. Electronics, 10(6), 668. https://doi.org/10.3390/electronics10060668

Rawal, S., Rawal, B., Shaheen, A., & Malik, S. (2017). Phishing detection in e-mails using machine learning. International Journal of Applied Information Systems, 12(7), 12-24. https://doi.org/10.5120/ijais2017451713

Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43905.pdf

Sarju, S., & Thomas, R. (2014). Spam Email Detection using Structural Features. International Journal of Computer Applications, 89(3). https://doi.org/10.5120/15485-4265

Shen, Y., He, X., Gao, J., Deng, L., & Mesnil, G. (2014, November). A latent semantic model with convolutional-pooling structure for information retrieval. In Proceedings of the 23rd ACM international conference on conference on information and knowledge management (pp. 101-110). https://doi.org/10.1145/2661829.2661935

Shin, H., & Paek, J. (2018). Automatic task classification via support vector machine and crowdsourcing. Mobile Information Systems, 2018. https://doi.org/10.1155/2018/6920679

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_campaign=buffer&utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com

Srivastava, P. (2017). Essentials of Deep Learning: Introduction to Long Short Term Memory. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

Thakur, H., & Kaur, S. (2016). A Survey Paper On Phishing Detection. International Journal of Advanced Research in Computer Science, 7(4). http://www.ijarcs.info/index.php/Ijarcs/article/view/2706

Tong, S., & Koller, D. (2001). Support vector machine active learning with applications to text classification. Journal of machine learning research, 2(Nov), 45-66. https://www.jmlr.org/papers/volume2/tong01a/tong01a.pdf

Vadivukarassi, M., Puviarasan, N., & Aruna, P. (2017). Sentimental analysis of tweets using Naive Bayes algorithm. World Applied Sciences Journal, 35(1), 54-59. https://www.academia.edu/download/55083588/7.pdf

Verma, P., Goyal, A., & Gigras, Y. (2020). Email phishing: Text classification using natural language processing. Computer Science and Information Technologies, 1(1), 1-12. https://doi.org/10.11591/csit.v1i1.p1-12

Xiang, G., Hong, J., Rose, C. P., & Cranor, L. (2011). Cantina+ a feature-rich machine learning framework for detecting phishing web sites. ACM Transactions on Information and System Security (TISSEC), 14(2), 1-28. https://doi.org/10.1145/2019599.2019606

Yang, Z., Qiao, C., Kan, W., & Qiu, J. (2019, April). Phishing Email Detection Based on Hybrid Features. In IOP Conference Series: Earth and Environmental Science (Vol. 252, No. 4, p. 042051). IOP Publishing. https://doi.org/10.1088/1755-1315/252/4/042051

Yasin, A., & Abuhasan, A. (2016). An intelligent classification model for phishing email detection. arXiv preprint arXiv:1608.02196. https://doi.org/10.5121/ijnsa.2016.8405

Yu, W. D., Nargundkar, S., & Tiruthani, N. (2009, July). Phishcatch-a phishing detection tool. In Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference-Volume 02 (pp. 451-456). https://doi.org/10.1109/COMPSAC.2009.175

Zhang, W., & Gao, F. (2011). An improvement to naive bayes for text classification. Procedia Engineering, 15, 2160-2164. https://doi.org/10.1016/j.proeng.2011.08.404

Zhang, X., Zeng, Y., Jin, X. B., Yan, Z. W., & Geng, G. G. (2017, December). Boosting the phishing detection performance by semantic analysis. In 2017 ieee international conference on big data (big data) (pp. 1063-1070). IEEE. https://doi.org/10.1109/BigData.2017.8258030

Zhang, Y., & Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820. https://arxiv.org/abs/1510.03820

Zhang, Y., Hong, J. I., & Cranor, L. F. (2007, May). Cantina: A content-based approach to detecting phishing web sites. In Proceedings of the 16th international conference on World Wide Web (pp. 639-648). https://doi.org/10.1145/1242572.1242659