

Catching the Phish: Detecting Phishing Attacks using Recurrent Neural Networks (RNNs)

Lukáš Halgáš¹, Ioannis Agraftotis¹, and Jason R.C. Nurse²

¹ Department of Computer Science, University of Oxford, United Kingdom
`{lukas.halgas,ioannis.agrafiotis}@cs.ox.ac.uk`

² School of Computing, University of Kent, Canterbury, United Kingdom
`j.r.c.nurse@kent.ac.uk`

Abstract. The emergence of online services in our daily lives has been accompanied by a range of malicious attempts to trick individuals into performing undesired actions, often to the benefit of the adversary. The most popular medium of these attempts is phishing attacks, mainly through emails and websites. In order to defend against such attacks, there is an urgent need for automated mechanisms to identify this malicious content before it reaches users. Machine learning techniques have gradually become the standard for such classification problems. However, identifying common measurable features of phishing content (e.g., in emails) is notoriously difficult. To address this problem, we engage in a novel study into a phishing content classifier based on a recurrent neural network (RNN), which identifies such features without human input. At this stage, we scope our research to emails, but our approach can be extended to apply to websites. Our results show that the proposed system outperforms state-of-the-art tools. Furthermore, our classifier is efficient and takes into account only the text and, in particular, the textual structure of the email. Since these features are rarely considered in email classification, we argue that our classifier can complement existing classifiers with high information gain.

Keywords: Phishing · Machine Learning · Recurrent Neural Networks · Natural Language Processing · Web Security.

1 Introduction

Advances in computer security have raised confidence in internet safety leading to e-commerce, internet banking and other means of sending, managing and receiving money online. Unfortunately, the advent of online services has been accompanied by illicit attempts to sham such transactions to the benefit of malicious entities. Perhaps the most popular and easy to execute attack, which poses a threat to organisations, institutions and simple users, is *phishing*.

Phishing is a type of cyber-attack that communicates socially engineered messages to humans using digital channels in order to persuade them to perform certain activities to the attacker’s benefit [12, 16]. Email is the most common avenue for a phishing attack, with almost 91% of successful cyber-attacks/security

breaches initiated by sending out spoofed emails [18]. The entire phishing operation can even be outsourced and automated [20], enabling the phishing threat to be, as it is, ubiquitous and continuous. Research has also found that it is increasingly difficult for humans to detect phishing attacks [10]. Therefore, there is a strong argument for automated mitigation methods to keep the user’s exposure to the attacks at a minimum.

The dynamic nature of phishing, with new trends and challenges constantly emerging, motivates a more adaptive filtering approach. Machine learning (ML) has been utilised, as the de-facto standard for classification purposes over many fields, email classification included. Developing a ML-based classifier to underline the phishing filtering is the approach we investigate in this paper. Our classifier analyses the text of the email and, in particular, the email’s language structure. It follows that our work is largely orthogonal to contemporary email classification systems which, to the best of the authors’ knowledge, do not employ natural language processing. We propose a novel detection system for phishing emails based on recurrent neural networks (RNNs). Our evaluation indicates that the RNN system outperforms state-of-the-art tools.

In what follows, Section 2 presents alternative approaches to automated detection of phishing emails and literature on current machine learning approaches. Section 3 details our methodology and feature selection for the RNN, while Section 4 describes the system implementation. Section 5 discusses the evaluation of the system, and Section 6 concludes the paper.

2 Current Landscape on Mitigation Techniques to Phishing

Techniques to mitigate the phishing problem include human training, laboriously curated blacklists and two-factor authentication for affected resources. However, to combat the problem of large numbers and the dynamic nature of phishing, automated techniques to detect new threats is required. Our work focuses on automated detection of phishing emails. We note that diverse techniques to combat phishing at various levels of the attack can be found in the literature. We reduce our treatment to the algorithmic classification of emails as phish or ham.

Specialised algorithms to classify email as phishing, spam (unsolicited email) or ham (i.e., not spam) have been the focus of research since the beginning of unsolicited email. Classification of phishing email is subsumed in the more general problem of spam filtering. As such, most email classifiers, hence filters, treat relatively harmless spam equivalently to dangerous phishing emails. We challenge the conventional parity in Subsection 3.3.

Phishing email and bulk spam email are distinct in intent and characteristics. While bulk spam uses sensationalistic language for advertising in personal inboxes, phishing emails mimic ham emails to raise confidence in its allegedly legitimate origin. Thus, Chandrasekaran et al. argue that filtering phishing emails needs to be treated separately from the bulk spam filtering [5]. Their classifier uses 23 *style marker features* including the number of words W , the number of characters C and

the number of words per character W/C alias as *vocabulary richness*, together with two structural attributes extracted from the email subject and body. The authors report results of up to perfect classification, with the accuracy dropping by 20% with the removal of the two structural features. Although the experiment used only a small corpus of 400 emails, the results demonstrate the importance of language and layout, or structure, of emails in phishing classification.

In practice, machine learning-based classifiers using a small set of characteristic features outperform those based on a broad set of general features. In the domain of phishing classification, for example, Fette et al. identified a subset of only ten features (of the hundreds of features commonly used to classify spam email), that best distinguish phishing emails from ham [6]. The resulting method outperformed the trained version of Apache’s SpamAssassin [22] classifier at identifying phishing emails, with the false negative rate reduced by a factor of ten. This result demonstrates that having specialised features for the task, in prominence to general email classification features, improves phishing classification.

Bergholz et al. build on top of this work by introducing advanced features for phishing classification [3]. The authors note that improvement in classification through variation of the classifying algorithm itself is statistically insignificant. In conclusion, statistically significant improvement is possible by inventing better features. Bergholz et al. hence develop two sets of advanced features based on unsupervised learning algorithms to complement 27 basic features commonly used in spam detection. One set of the advanced features are based on a dynamic Markov chain (DMC) language model, and additional word-clusters, or email topics, are based on a latent Dirichlet allocation (LDA) model. The best results occur when the advanced features are used in conjunction with the basic features, achieving the state-of-the-art [3].

Toolan et al. analysed 40 basic features popularly used in email classification and ranked them based on their *information gain* to the classification task at hand [24]. The most informative features were vocabulary richness of the email body and the subject. Other popular features performed very poorly, indicating that our intuitive understanding of what constitutes a phishing email may be very wrong. This is illustrated by the failure to gain information from counting <form> elements or finding the word ‘debit’ in the subject. We may attribute the results to a shift in phishing trends, or to the failure of human experts to identify useful features. The authors also conclude that language modelling approaches to phishing classification are the most promising.

In line with the results and conclusions of previous work, we algorithmically design an advanced feature based on little human intuition. In particular, we train a recurrent neural network (RNN) as a language model to distinguish phishing from ham based on the text of the email only. We describe our advanced feature in detail in Section 4.

3 Methodology

The *classification* task is to identify which of a finite number k of categories, or classes $C = \{c_1, \dots, c_k\}$, a sample \mathbf{x} belongs to, i.e., deduce a classifier or mapping, $\mathbf{x} \mapsto c$. In our application to phishing, the classification is a mapping of email representations to the label set $\{phish, ham\}$.

The machine learning (ML) approach to classification is to automatically establish a function f that determines the desired class

$$\hat{y} = f(\mathbf{x}) \in \{ham, phish\}$$

on the input of a representation \mathbf{x} of an email. The function f is parameterised by values θ . During the training phase, the parameter values θ are determined to reproduce a relation between the input \mathbf{x} and class label y in agreement with a training set $\{(\mathbf{x}_0, y_0), \dots, (\mathbf{x}_n, y_n)\}$ of pre-classified samples and a suitable optimisation criterion. In this sense, the ML approach is to extrapolate the relationship between the observed sample points and class labels to unlabelled input \mathbf{x} and its predicted class $\hat{y} = f(\mathbf{x})$.

3.1 Feature Identification

An input \mathbf{x}_{raw} representing an email as a (very long) series of binary digits, comprising the raw source code of an email in binary format, is unwieldy for an algorithm to detect patterns. We hence use a more compact representation of the input as a feature vector $\mathbf{x} = (f_1(\mathbf{x}_{\text{raw}}), \dots, f_m(\mathbf{x}_{\text{raw}}))$. Features should characterise an email with respect to the current classification problem. The relative inaccuracy of ML-based spam classifiers on the seemingly similar task of phishing classification illustrates the need for specialised features for this task [6].

Features are most often identified by experts, in line with their intuitive understanding of “phishiness” or “hamness”. Toolan et al. [24] demonstrated that such intuitively sound features often fail to be relevant in phishing classification. On the other hand, structural features have empirically been indicative of emails being ham or phish [6]. Language modelling approaches to improving the state-of-the-art in phishing classification are considered the most worthwhile to research [24]. Naturally, we follow the language modelling approach to design a strong feature based on the structure and content of the email’s body.

Natural language processing (NLP) is the field of Computer Science studying human-machine interactions and, in particular, establishing and exploiting language models. The rich structure and ambiguity of natural languages make it difficult to identify and extract complex language features, such as the tone of urgency in the email body. Previously, Verma et al. used pre-trained WordNet hypernymy trees of sets of words conveying urgency or action, among other characteristics, to identify sentences and hence emails as actionable or informative [25].

In the unsupervised learning approach, the ML algorithm detects data patterns in the dataset without supervision or specific expert advice. That is, the training of

the model determines, or learns, the features itself. Fortunately, the unsupervised learning algorithms form the state-of-the-art techniques in language modelling. It follows that our approach of training a language model as a feature is not susceptible to mistakes in expert feature identification.

3.2 Deep Learning

Neural networks (NNs) are a computational model, in the quintessential example of a multilayer perceptron resembling a hierarchical network of units, or neurones. The hierarchical structure intuitively gives NNs the capacity to extract high-level features from simple data, i.e., to disentangle and winnow the factor of variation in the NN's input. This intuition of NN structure makes NNs suitable for the task of *representation learning*, or automatic feature identification.

Recurrent neural networks (RNNs), the deepest of all learners, are a family of NNs specialised for processing sequential data. Like Markov chain models, RNNs have the advantage of processing data in sequence, thus accounting for the order of data. The input text is usually abstracted to a sequence of characters, words or phrases. Undoubtedly, the order of words is valuable in language modelling. RNNs form the backbone of the current state-of-the-art language models, so an RNN language model should form an accurate content-based classifier of emails.

The literature presents several applications of NNs to the related problem of classifying malicious URLs and websites [1, 14, 26, 28]. Bahnsen et al. have used RNNs to classify web addresses as linking to phishing or legitimate websites [1]. The input was a sequence of the URL's characters only. Similarly, we use RNNs to classify emails as phishing or ham, with the input being a sequence of words only.

We alleviate the learning problem from language modelling to the binary classification of email to phish or ham. This classification can be trivially abstracted to predicting y , where $y = 1$ if *phish* and $y = 0$ if *ham*. We thus get a supervised learning problem with representation learning. This simpler task overcomes the often-prohibitive computational cost of training a full-blown language model. Inherently, the RNN classifier models a $y \sim \text{Bernoulli}(p_{\mathbf{x}})$ distribution using a sigmoid output unit

$$p_{\mathbf{x}} = \sigma(z_{\mathbf{x}}) := \frac{1}{1 + \exp(-z_{\mathbf{x}})} = \frac{\exp(z_{\mathbf{x}})}{\sum_{y'=0}^1 \exp(y' z_{\mathbf{x}})}$$

where $z_{\mathbf{x}}$ is the output of the last linear layer, dependent on the RNN input \mathbf{x} . Intuitively, this is the normalisation of the unnormalised probability distribution

$$\begin{aligned} \tilde{p}_{\mathbf{x}}(c) &= \exp(c z_{\mathbf{x}}) \\ \log \tilde{p}_{\mathbf{x}}(c) &= c z_{\mathbf{x}} \end{aligned}$$

for $c \in \{0, 1\}$. Then $p_{\mathbf{x}} = p(y = 1 \mid \text{sequence of words of email } \mathbf{x}) \in [0, 1]$ gives the email label prediction $\hat{y} = \arg \max_{c \in \{0, 1\}} p(y = c \mid \mathbf{x}) = \mathbb{1}\{z_{\mathbf{x}} \geq 0\}$.

3.3 Precision/Accuracy Trade-off

Commonly, misclassification of phishing and ham emails are considered to have different weights of error [4,6]. A false positive, or incorrectly labelling a legitimate email as phishing is considered to be a more severe error than a false negative, or misclassifying a phishing email as ham. This convention is an artefact of bulk spam filtering, where being exposed to harmless unsolicited advertising is smaller harm than having a legitimate email undelivered. Note that phishing filtering is a subproblem of spam filtering, with techniques as well as related conventions adapted from the more general problem.

It follows that many phishing classifiers emphasise the *precision* of the classifier together with its *accuracy* as criteria of classifier merit. However, a phishing email generally poses more harm than a bulk spam email. Despite the common practice in related work, we do not introduce different penalties to the various misclassification.

Nevertheless, our technique, as most ML-based techniques, is flexible to the trade-off of between accuracy and precision. The RNN classifier, described above, gives a probability value $p(y = 1 \mid \mathbf{x}) \in [0, 1]$ of the email \mathbf{x} being phish. This output value can be viewed as the confidence of the email being phish, where a value close to 0.5 means low confidence in the classification. Thus, a simple method to increase precision, at the cost of lowering accuracy, is to increase the threshold value for labelling an email phish.

4 Design and Implementation

Our RNN classifier labels an input email as either a legitimate email or a phishing attempt. In this section, we describe the procedure of transforming the raw email source into a variable size vector of integers that is input to the RNN itself. The preprocessing of the emails is illustrated in Fig. 1. Use of the trained RNN classifier, also illustrated in the figure, is described in Section 5.

4.1 Preprocessing for the RNN Classifier

Our binary classification RNN model takes sequences of integer values as input and outputs a value between 0 and 1. We abstract the computer-native copy of an email as a sequence of bytes into the high-level representation as a sequence of symbol and word tokens, represented as unique integers. It is customary to ‘feed’ RNNs with an n -gram representation of the abstracted text. Due to the small size of our dataset, our dictionary of n -grams would contain very few repetitive phrases of n words for values $n \geq 2$. For the balance of token expressiveness, and vocabulary size, we choose to represent emails as sequences of 1-grams, or single-word tokens.

Note that our classifier only considers the text of emails in making its classification decision. Thus, useful features, such as those based on linked web address analysis, are entirely orthogonal to our classifier and thus are largely complementary. As an initial step in preprocessing of the classified email, we extract its text in plaintext format.

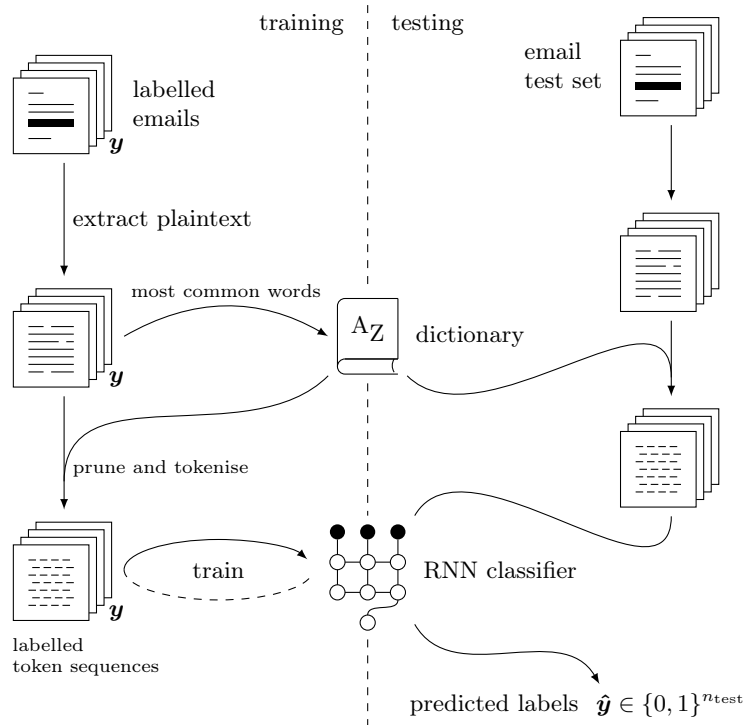


Fig. 1. Training and testing pipeline. We establish a dictionary of the most common words and train our recurrent neural network (RNN) classifier during the training phase. Preprocessing of the emails input into the RNN consists of extracting plaintext, which is then pruned and transformed into a sequence of tokens of manageable size.

4.2 Tokenising the Text

We seek flexibility in tokenising the text through fine-tuning the parameters of the tokeniser, such as rules of what word or character sequences to represent by the same token. The naïve approach of splitting on whitespace characters does not generalise well to email tokenising. Incautious or malicious salting, e.g., inconsistent whitespace or the ubiquity of special characters, form words unique to an email. Considering such tokens would inherently lead to overfitting, based on the presence of unique traits.

Our approach to tokenising is that of adjusted word-splitting. First, we lowercase all characters in the email and remove all characters the RFC 3986 standard does not allow to be present in a URL, i.e., we only keep the unreserved `a-z`, `0-9`, `-`, `.`, `_` and reserved `:/?#[]@!$&'()*+,-;=` characters and the percentage sign `%`. Although this step is motivated by the ease of later identifying URLs for the `<url>` token determination, we get the benefit of restricting our character base cardinality to 61. The 60th character, which RFC 3986 does not allow in URL, but we do not immediately replace with whitespace, is the quote character `"`, which is often used in emails. Note, the 61st character is the whitespace character.

We introduce four special tokens summed up in Table 1, and, nine tokens for the special characters left, replacing dots, quotes and seven other special characters with their respective tokens. Finally, we split the clean text into words, serving as their individual tokens, and prepend and append the start `<s>` and end `<e>` tokens, respectively, to the tidy sequence of tokens.

<code><url></code>	replaces a URL beginning with <code>http://</code> or <code>https://</code> ,
<code><www></code>	replaces a URL beginning with the informal <code>www.</code> ,
<code><email></code>	stands for an email address,
<code><threespecial></code>	groups together and replaces three or more consecutive non-alphanumeric characters, possibly separated by whitespace,
<code><s></code>	beginning, or start, of email,
<code><e></code>	end of email.

Table 1. As a preprocessing step, we abstract out web addresses and email addresses, as well as longer sequences of non-alphanumeric characters. We replace them with special tokens, summarised in the Table.

The final representation of the email includes only lowercase alphanumeric words and tokens. Using a list of allowed characters, we aggressively parse the text, mitigating the threat of the text exhibiting unexpected behaviour.

4.3 Recurrent Neural Network Classifier

Our model is a simple RNN, consisting of an encoding layer, two recurrent layers, and a linear output layer with a Softplus activation, as shown in Fig. 2. Challenges of training deep networks, of which RNNs are the deepest, motivate most of the design decisions presented in this section.

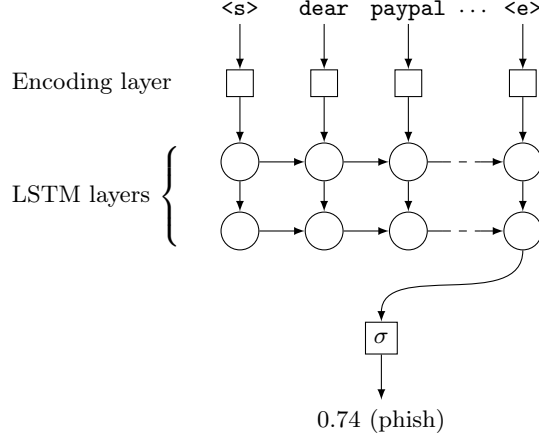


Fig. 2. Our recurrent neural network (RNN) classifier: The input is a sequence of tokens, each mapped to a learned vector representation. The output is a value between 0 and 1, with a value over 0.5 being classified as phish.

We implement our recurrent layers with the long short-term memory (LSTM) architecture [9]. LSTM is a gated recurrent neural layer architecture that, through its carefully designed self-loops, can learn long-range dependencies. We use a variation of the original concept with weights on the self-loop conditioned on the context [7]. Due to its carefully crafted architecture, LSTMs are resistant to the vanishing gradient problem [2]. As is the standard, we use the tanh nonlinear activation on the cells' output. We describe the choice of the size of the hidden layer in Subsection 5.1, but we will choose the hidden state to be 200 variables large.

The output \mathbf{h}_2 of the last LSTM cell of the second layer is input further up the model. So that our model outputs a single variable $p_{\hat{y}} \in (0, 1)$ as required. Since we are modelling a Bernoulli probability, we use the simplest linear layer

$$\mathbf{h}_2 \mapsto \mathbf{w}^\top \mathbf{h}_2 + b = z,$$

consisting of a weight vector \mathbf{w} and bias scalar b . The final output is obtained by mapping the linear layer output scalar through the logistic sigmoid function

$$p_{\hat{y}} = \sigma(z) := \frac{1}{1 + \exp(-z)} \in (0, 1)$$

to obtain the estimated probability of an email being phish.

4.4 Input Sequence Preprocessing

If we let every token in the dataset to have its unique embedding vector, not only would the encoding layer be huge, but our model predictions would not generalise well to any emails containing unknown words. We hence reduce the size of the *dictionary* considered by our model, in order to acquire round values, to the 4 989 most common words in the training and validation sets of emails as token sets. In other words, we do not consider repetitions of a word in a single email in determining the occurrence count.

Every token in the dictionary is assigned a unique index value. So that our vocabulary reduction is not too harsh, we unite tokens of similar meaning. We stem the words using the Snowball Stemmer, a more aggressive version of the popular Porter Stemmer [19]. We then add five more tokens, `<unkalpha>`, `<unknum>`, `<unk>`, `<cuts>` and `<cute>`, to the dictionary. We summarise the new tokens in Table 2. The first three abstract out unknown words to the dictionary, such as those that consist of only alphabetical or numerical values, or fit none of the first two, respectively. We describe the final two tokens in Section 4.5 below. Note that the eleven special tokens described in Tables 1 and 2, together with the 4 989 most common word tokens form the complete dictionary of 5 000 tokens. The number is arbitrary.

<code><unkalpha></code>	words of alphabetical characters not in dictionary,
<code><unknum></code>	numbers not in dictionary,
<code><unk></code>	other unknown words,
<code><cuts></code>	beginning, or start, of a pruning cut,
<code><cute></code>	end of a pruning cut.

Table 2. To make the set of tokens manageably small, we only keep the most common words and replace the rest with special tokens indicating that they are unknown to our dictionary. To make the size of the email representation manageable, we limit the size of emails to 1 000 tokens and replace the parts cut out with special tokens.

4.5 Cutout Pruning

Anomalous emails of very long sequence representations cause training inefficiency, amongst other problems, in evaluating very long-range dependencies. The problem is that such long emails cause unnecessary ‘padding’ of other, shorter sequences, when employing gradient-based learning in batches, reducing stability and the speed of learning. Most notably, modern GPU architectures take time proportional

to the maximum length of a sample in the batch to evaluate batched samples, as we do.

We hence compromise our email representation for excessively long emails via a simple pruning procedure. The idea is to cut out a sequence of size a third of our threshold of 1 000 tokens, and ‘glue’ the beginning and end of the email to the *cutout* sequence. The concept is to keep the beginning, most middle and ending parts of the email, skipping the uninformative bits of ham or phish emails. To allow our model to grasp the idea of the anomaly introduced in close-neighbour word dependencies, we add two tokens, <cuts> and <cute>, to the dictionary to represent a start or an end of a sequence caused by the pruning cut. Intuitively, we think of these tokens as ‘glue’.

Emails represented as sequences of indices of their respective tokens, in the range of the dictionary size $V = 5000$, are input or ‘fed’ to the RNN. The first, *encoding* layer, encodes each index in sequence with its corresponding token embedding. The embedding vectors elements are initialised as random Gaussian $\mathcal{N}(0, 0.1^2)$ values and learned as parameters of the model.

5 Evaluation

Before presenting the results of our RNN classifier, we first introduce the email datasets used in the evaluation. Table 3 presents a summary of the datasets used. The first dataset, **SA-JN**, is a combination of all 6 951 ham emails from the SpamAssassin public corpus [22] and 4 572 phishing emails from the Nazario phishing corpus [15] collected before August 2007. **SA-JN** is a accessible dataset used in related work to evaluate comparable phishing detection solutions [3, 6, 25].

Our second dataset, **En-JN**, is a combination of the Enron email dataset combined with phishing emails from the Nazario phishing corpus. The Enron email dataset is generated by 158 employees of the Enron Corporation, and, to the best of the authors’ knowledge, is the only large public dataset of real-world emails. We combine a randomly selected subset of 10 000 emails from the Enron dataset together with all 9 962 phishing emails from the Nazario phishing corpus.

Corpus	Size	Ham	Phishing	Source
SA-JN	11 523	6 951 (60%)	4 572 (40%)	SpamAssassin and Nazario
En-JN	19 962	10 000 (50%)	9 962 (50%)	Enron and Nazario

Table 3. Decomposition of datasets used in evaluation.

As is common practice in statistical learning, we split the data samples for training and evaluation. Separately, we sort the ham and phishing emails by the `datetime` stamp extracted from the email `Received` or `Received-Date` field (defined to be the maximum, or latest, timestamp where multiple `Received` or `Received-Date` fields are present). Consequently, we get two sorted lists, that

we separately split into *training and validation*, and *testing* sets, with a 9-1 ratio twice. The respective 81% – 9% – 10% splits respect the received datetime stamps with the most recent 10% of the emails forming the testing set. The underlying reasoning is to approximate the real scenario of training the classifier on present data to predict future data. We then combine the ham and phishing sets, respecting the splits.

We evaluate our classifier against the most popular metrics in email classifications, which we introduce shortly. We then compare our language model to other content-based classifiers.

5.1 Training

The encoding itself accounts for $5000 \times 200 = 1$ mil parameters of the model. The challenge of training so many parameters of a network requires more advanced optimisation algorithms. We employ the following techniques for optimisation and regularisation of our model.

We initialise the weights of the LSTM cells to random orthogonal matrices with the gain set to 5/3 for the weights of the cell gate with tanh activations, and set the other weights, with sigmoid activations, to orthogonal matrices with gain 1 [21]. It is the perfect orthogonality of the weight matrices that motivated our choice for the embedding and LSTM to share the same unit size of 200.

As suggested by Jozefowicz et al. [11], we initialise the bias of the LSTM forget gate to 1, and initialise all other biases to 0 throughout the RNN. We initialise the weights outside of the recurrent layers by sampling from the Gaussian $\mathcal{N}(0, 0.1^2)$ distribution. The model contains dropout [23] of 0.2 on the embedding layer, a dropout of 0.5 between all recurrent states on top of each other, with no dropout in-between successive states of a recurrent layer, as proposed by Sutskever et al. [27]. We also add dropout of 0.5 at the final output of the recurrent layer.

The model is optimised using the Adam optimiser [13] against the binary cross-entropy loss function. We train the model with batches of size 200 samples. We shuffle the training dataset at the beginning of every epoch. To tackle the exploding gradient problem, we clip the gradient norm $\|\mathbf{g}\|$ [17] with threshold 1. Finally, we stop training early with continuation of learning [8] by training over the validation set once.

5.2 Evaluation Metrics

Given that the datasets used for email classification vary significantly in how even their distributions are, the apparent *accuracy* measure is of limited value for comparison to other classifiers. We hence report the standard measures of *precision*, *recall*, the *F-measure*, *false positive* and *false negative rates* in addition to accuracy.

We note that email classification errors vary in importance. As an artefact of the problem of spam email classification, it is common practice to consider

a false positive error to be more costly than a false negative misclassification. However, this is under the assumption of aggressive filtering of positives and harmless false negatives. In the domain of phishing emails, however, false negatives present significant danger and less aggressive filtering methods such as alerts and link-disabling are common.

We train the classifier over four epochs on the training dataset and one more epoch over the validation dataset. Because the model is expensive to train, in time and computational power, the results provided are of the single trained instance. We evaluate the model on the test set, which had been unseen during training, and is chronologically separated from training a validation set. This is because we split each dataset into training, validation and testing sets in chronological order.

Our classifier is most directly comparable to other text-based features, or sub-classifiers that analyse the text of the classified email only. We compare our work with the textAnalysis sub-classifier of the PhishNet-NLP email classifier by Verma et al. [25], and the state-of-the-art dynamic Markov chain (DMC) model proposed by Bergholz et al. [3]. We summarise the results in Table 4.

	corpus	accuracy	fp-rate	fn-rate	precision	recall	<i>F</i> -measure
textAnalysis	?-JN	78.54 %	14.90 %	22.90 %	95.93 %	77.10 %	85.49 %
DMC _{text}	SA-JN	99.56 %	0.00 %	4.02 %	100.00 %	95.98 %	97.95 %
our RNN	SA-JN	98.91 %	1.26 %	1.47 %	98.74 %	98.53 %	98.63 %
our RNN	En-JN	96.74 %	2.50 %	4.02 %	97.45 %	95.98 %	96.71 %

Table 4. Summary of our results in comparison to related work in popular metrics.

Our test dataset is well-separated from the training set. We could argue that the classification problem we evaluated our classifier against is unrealistically hard. Intuitively, messages arriving in a specific inbox would exhibit more pronounced patterns, and would thus be easier to classify correctly.

Verma et al. [25] propose that textAnalysis offers a classification value very independent from the other features, as only the text of the email is considered. For the same reason, our classifier should not copy the labels of other features present in classification, but rather provide an independent view on the classification at hand.

The RNN classifier outperforms the textAnalysis classifier and has comparable results to the state-of-the-art DMC_{text} feature. We note that perfect classification is not possible in our setting, as two emails with the same token sequence will necessarily be labelled equally. Since both, ham and phishing email corpora contain empty emails with attachments, which have been removed, the emails are identical to our classifier. This proves inseparability of the emails with the word-sequence representation.

6 Conclusion

In this paper, we propose a novel automated system aiming to mitigate the threat of phishing emails with the use of RNNs. Our results suggest that the flexibility of RNNs gives our system an edge over the expert feature selection procedure, which is vastly employed in ML-based attempts at phishing mitigation.

We focused on the overlooked content source of email information and demonstrated its utility when considered in phishing threat mitigation. The nature of RNN and its training procedure make it suitable for the case of online learning deployment. Our classifier could theoretically change over time to capture new trends continuously and keep up accurate and precise classification throughout. Our results have demonstrated a wealth of potential in non-trivial feature identification for classifying emails since our system's performance surpasses the state-of-the-art systems which are based on features designed by human intuition.

Finally, it is worth noting that the general criticism of supervised learning extends to our case. Little information is provided by the RNN classifier on the nature of emails at successful classification. The proposed solution generalises easily to the case of inclusion of basic spam email, and is a prospect for further automated success.

References

1. Bahnsen, A.C., Bohorquez, E.C., Villegas, S., Vargas, J., González, F.A.: Classifying phishing URLs using recurrent neural networks. In: Proceedings of the APWG Symposium on Electronic Crime Research. eCrime '17, IEEE (April 2017). <https://doi.org/10.1109/ECRIME.2017.7945048>
2. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* **5**(2), 157–166 (March 1994). <https://doi.org/10.1109/72.279181>
3. Bergholz, A., Chang, J.H., Paaß, G., Reichartz, F., Strobel, S.: Improved phishing detection using model-based features. In: Proceedings of the Fifth Conference on Email and Anti-Spam. CEAS '08 (August 2008)
4. Bergholz, A., De Beer, J., Glahn, S., Moens, M.F., Paaß, G., Strobel, S.: New filtering approaches for phishing email. *Journal of Computer Security – special issue on EU-funded ICT research on Trust and Security* **18**(1), 7–35 (January 2010). <https://doi.org/10.3233/JCS-2010-0371>
5. Chandrasekaran, M., Narayanan, K., Upadhyaya, S.: Phishing email detection based on structural properties. In: Proceedings of the 9th Annual NYS Cyber Security Conference. NYSCSC '06 (June 2006)
6. Fette, I., Sadeh, N., Tomasic, A.: Learning to detect phishing emails. In: Proceedings of the 16th international conference on World Wide Web. pp. 649–656. WWW '07, ACM (May 2007). <https://doi.org/10.1145/1242572.1242660>
7. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm. *Neural Computation* **12**(10), 2451–2471 (October 2000). <https://doi.org/10.1109/72.279181>
8. Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*. MIT Press, Cambridge, MA, USA (2016), <https://www.deeplearningbook.org>

9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Computation* **9**(8), 1735–1780 (November 1997). <https://doi.org/10.1162/neco.1997.9.8.1735>
10. Iuga, C., Nurse, J.R.C., Erola, A.: Baiting the hook: Factors impacting susceptibility to phishing attacks. *Human-centric Computing and Information Sciences* **6** (June 2016). <https://doi.org/10.1186/s13673-016-0065-2>
11. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: *Proceedings of the 32nd International Conference on Machine Learning*. pp. 2342–2350. ICML '15 (July 2015)
12. Khonji, M., Iraqi, Y., Jones, A.: Phishing detection: A literature survey. *IEEE Communications Surveys & Tutorials* **15**(4), 2091–2121 (April 2013). <https://doi.org/10.1109/SURV.2013.032213.00009>
13. Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. *arXiv preprint* (2014), <https://arxiv.org/abs/1412.6980>
14. Mohammad, R.M., Thabtah, F., McCluskey, L.: Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications* **25**(2), 443–458 (August 2014). <https://doi.org/10.1007/s00521-013-1490-z>
15. Nazario, J.: <https://monkey.org/~jose/phishing/>
16. Nurse, J.R.C.: Cybercrime and you: How criminals attack and the human factors that they seek to exploit. In: *The Oxford Handbook of Cyberpsychology*. Oxford University Press, Oxford, UK (May 2019). <https://doi.org/10.1093/oxfordhb/9780198812746.013.35>
17. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. *arXiv preprint* (2012), <https://arxiv.org/abs/1211.5063>
18. PhishMe, Inc.: 2016 enterprise phishing susceptibility and resiliency report (2016)
19. Porter, M.F.: Snowball: A language for stemming algorithms, <https://snowballstem.org/>
20. Ramzan, Z.: Phishing attacks and countermeasures. *Handbook of Information and Communication Security* pp. 433–448 (2010). https://doi.org/10.1007/978-3-642-04117-4_23
21. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint* (2013), <https://arxiv.org/abs/1312.6120>
22. SpamAssassin: <https://spamassassin.apache.org/old/publiccorpus/>
23. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**(1), 1929–1958 (January 2014)
24. Toolan, F., Carthy, J.: Feature selection for spam and phishing detection. In: *2010 eCrime Researchers Summit*. pp. 1–12. IEEE (October 2010). <https://doi.org/10.1109/ecrime.2010.5706696>
25. Verma, R., Shashidhar, N., Hossain, N.: Detecting phishing emails the natural language way. In: *17th European Symposium on Research in Computer Security*. pp. 824–841. ESORICS '12, Springer, Berlin, Heidelberg (September 2012). https://doi.org/10.1007/978-3-642-33167-1_47
26. Vinayakumar, R., Soman, K.P., Poornachandran, P.: Evaluating deep learning approaches to characterize and classify malicious URLs. *Journal of Intelligent & Fuzzy Systems* **34**(3), 1333–1343 (March 2018). <https://doi.org/10.3233/JIFS-169423>
27. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. *arXiv preprint* (2014), <https://arxiv.org/abs/arXiv:1409.2329>

28. Zhao, J., Wang, N., Ma, Q., Cheng, Z.: Classifying malicious URLs using gated recurrent neural networks. In: Proceedings of the 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 385–394. IMIS '18, Springer, Cham (July 2018). https://doi.org/10.1007/978-3-319-93554-6_36