978-1-5090-3525-0/16/$31.00 ©2016 IEEE

Using Natural Language Processing Models for Understanding Network Anomalies

Ketul Barot Jialing Zhang Seung Woo Son
University of Massachusetts Lowell
Email: seungwoo_son@uml.edu

***Abstract*—This paper presents a case study where Natural Language Processing (NLP) techniques are applied to non-textual data. Traditionally NLP is applied to text data as it is meant for natural language. In this study, we explore the effects of NLP on non-textual data. Our motivation behind this study is that if the training data for the NLP algorithm meets proper semantic and syntactic analysis of the algorithm, it would provide meaningful interpretation for the non-textual data just as it works for text data. Specifically, we apply word2vec, a neural embedding model developed by Google, to detect types of attacks in network log data. Our proposed mechanisms rely on machine learning techniques for studying anomalies in network log data. We first describe how word2vec works and how it proves to be useful to find the semantic of a particular network attack types, which then can be used by other classification based machine learning algorithms, like Convolutional Neural Network (CNN).**

1. Introduction

Natural Language Processing (NLP) is mainly concerned with how computers interpret human (natural) languages. Modern NLP algorithms are based on machine learning, and earlier implementation of NLP involved manually writing a large set of rules. The machine learning algorithms automatically learn such rules by training on large corpora of real world examples. A corpus is a set of documents or sometimes individual sentences that have been annotated with correct labels to be learned. NLP is used in many Big Data problems, such as automatic summarization, sentiment analysis, question answering, anomaly detection in text data and other such applications.

In this paper, NLP is explored in detecting network anomaly. Network anomaly detection is part of a broader category of system known as network intrusion detection systems. Most of the network intrusion detection systems incorporate two main techniques: anomaly detection, which involves detecting deviations from normal behavior, and signature detection, which looks into anomaly or attack signatures and known intrusion detection signatures. Given that the cyber security landscape is evolving constantly, sophisticated malware are increasingly being developed along with the advancement in intrusion technology. For example, according to 2015 annual report from PandaLabs [1], more than 84 million new attack samples were detected and neutralized by PandaLabs, with an average of 230,000 samples daily. Most of these malware are known variants of existing malware to avoid detection from the current security products like anti-virus, firewalls, and other intrusion detection systems.

By definition, anomaly detection is the identification of items, events or observations which do not conform to an expected pattern [2]. Anomaly detection techniques can be categorized into three broad categories, namely, unsupervised anomaly detection, supervised anomaly detection, and semi-supervised anomaly detection. In the unsupervised anomaly detection, anomalies are detected in unlabelled test data, while in the supervised anomaly detection, anomalies are detected in labeled test data that describe either the data are normal or abnormal. In the semi-supervised anomaly detection, a small set of data is first trained to learn both normal and abnormal behavior, and then it tests for likelihood of a test instance to be generated by the learned model.

However, accurately detecting the anomalies can be very difficult for several reasons. First of all, the quantity that can be classified as anomaly always increases. Also, the application system intended to find anomalies is constantly evolving, therefore the detection scheme should be efficient enough to detect the subtle changes in the patterns early enough to prevent any damage to the system. Furthermore, because anomalies are unexpected in nature, it becomes more difficult to build an effective detection system. Therefore, there is a constant need to improve detection systems and keep them up-to-date to detect even subtle changes in a normal functioning network.

To address aforementioned problems, in this study, we explore NLP on non-textual data. Our motivation behind this is that if the sampled train data meets proper semantic and syntactic requirements of the NLP algorithm, the algorithm would provide useful interpretation on the non-textual data as well. Numerous works have been done in network anomaly detection field using machine learning algorithms like PCA for intrusion detection [3], clustering techniques [4], and also some hybrid techniques which combine different machine learning algorithms to detect the anomaly in network data.

A tutorial from Kaggle, "Bag of words meet Bag of Popcorn" [5], particularly motivated us to use a NLP algorithm on non-textual data. Their goal was to perform sentiment analysis on movie reviews where the positive and negative reviews were classified from the training dataset so the system automatically identifies the sentiment of the reviews. In this paper, we argue that the same concept could

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

be applied to our target domain, i.e., classifying normal and abnormal network data. Abnormal network data can be a data sample with different types of network attack, such as denial of service attack, probing attack, or user-to-remote attack [6], [7], etc.

We use word2vec [8], an NLP algorithm developed by Google, which is a neural network based method that converts words to vector representation in such a way that semantically and syntactically similar words are closely positioned in the space. These vectors can then be further used as features in deep neural network models such as convolutional neural networks (CNN) and recurrent neural networks (RNN) for various applications, for example, language modeling, sentiment analysis, etc. In the following section, we discuss how we train the network log dataset with word2vec to study the anomalies in the data. Our experimental results show that we successfully identify 85% of all the factors responsible for a particular attack (abnormal data).

2. Background

NLP generally involves training on text data. Typically, NLP algorithms use a sample input data set with words or sentences arranged in a particular format. The algorithm then extracts the features out of this training data. More often some other machine learning algorithms are used in addition to NLP algorithms in order to exploit these features and learn classification rules. After training, the machine learning algorithm predicts the output on a test data. Accuracy of classifier can be highly improved if larger training data set is used. Given this, the effectiveness of NLP largely depends on the training data in order to achieve a desired result. The remainder of this section is dedicated to the discussion of details using word2vec for anomaly detection as we use word2vec to generate features for anomaly detection.

Word2vec is an algorithm that takes a word as the input and produces its equivalent vector representation as the output. More specifically, it first builds a vocabulary consisting of unique words from the training text data and then generates vector representation of words in vocabulary. Internal vector conversion is done using two models, continuous-bag-of-words (CBOW) or skip-gram (SG). These two models use neural networks, which are trained to reconstruct syntactic and semantic context of words. Given a word, the network is trained to predict contexts (adjacent words in an input text) [9].

The easiest way to understand the vector representation is finding the nearest words for a specified input word. There is a script available in the word2vec toolkit called "distance", which calculates the cosine similarity between the input word and other words in the vocabulary. For example, if you enter "india" it will emit the similar words and their cosine distances with "india" [8]. The cosine distance value is between 0 and 1. The closer to 1, higher the output word is related to the input word. The actual outcome from word2vec is shown in Figure 1.

Word	Cosine distance
pakistan	0.681234
bangladesh	0.679523
bhutan	0.674235
myanmar	0.670012
nepal	0.669825
srilanka	0.654586
china	0.648579

Figure 1. Word Cosine Distance when input "india" is given to the trained vocabulary.

The word2vec is a neural network model that learns vector representation of words in the training file. A neural network is defined as "a highly interconnected processing computing system, which process information by their dynamic state response to external inputs" [10], basically meaning that neural networks are organized in layers. There are three layers involved namely: input, hidden, and output layers. The features are fed to the input layer. The input layer then communicates with a hidden layer, which performs the actual processing using a system of weighted connections. Finally, a output layer gives the output to the given input pattern. All the three layers are made up of a number of interconnected nodes, which contain an activation function [11].

We next discuss how the two models (CBOW and Skip gram) work in detail.

2.1. CBOW

To understand continuous bag of words (CBOW), we first need to understand what is bag of words (BOW). Consider the vocabulary of the training word vectors as V and N be the dimension of the vectors, then the input to the hidden layer would be a matrix WI of size $V \times N$, where each row represents a word from the vocabulary. In similar way, hidden layer would connect to output layer having a matrix WO, which would be of size $N \times V$.

Let us consider a training sample of following sentences: "the cat saw a mouse", "the cat ran after mouse", "the mouse went inside hole". The corpus vocabulary has nine unique words. For this example, the neural network will have nine input neurons and nine output neurons. If we decide to use three neurons in the hidden layer, then our matrices would be $WI = 9 \times 3$ and $WO = 3 \times 9$. Let us assume WI and WO be initialized to following values:

```
WI =
-0.094912   -0.425354    0.314589
-0.490589   -0.292356    0.056604
0.0492712    0.172246   -0.537715
 0.014528   -0.562333    0.089564
-0.337080   -0.156495   -0.034282
 0.460511   -0.129749   -0.414299
 0.115174    0.099682    0.366564
-0.044224    0.417952    0.326310
 0.256789    0.356489   -0.554896

WO =
  0.24589     0.57894    0.00489   -0.254689   -0.458213
 0.087945    0.021354   0.657892     0.00245
  0.01548     0.68791    0.02358     0.35678   -0.011554
-0.004589   -0.568974   0.002589     0.27894
 -0.42518     0.78945   -0.85479    0.015127    0.122489
 -0.05789     0.09785   0.258967    0.574859
```

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

Suppose we want to learn the relationship between "mouse" and "hole". In the word embedding terminology, the word "mouse" is called a context word and the word "hole" is called as a target word. In this case, the input vector X would be $[000010000]^T$, where only the fifth position is 1 due to the alphabetical sorting of the vocabulary. So the word "mouse" takes fifth position and similarly the target word "hole" would be $[00100000]^T$. With "mouse" as the input vector, the hidden layer neuron can be computed as: $H^T = X^T WI = [-0.337080 \quad -0.156495 \quad -0.034282]$.

The vector H of hidden neuron output mimics the second row of the input layer, as the input word is "mouse" which holds fourth position in the vocabulary. Similarly, carrying out matrix multiplication from hidden layer to output layer, the vector for output neurons can be calculated as: $H^T WO$ = [-0.07073, -0.32986, 0.02396, 0.02949, 0.15206, -0.02694, 0.07848, -0.23104, -0.06418].

Thus the probabilities for words in the output layer can be calculated by converting activation values to probabilities. To calculate this, word2vec incorporates a gradient log normalizer function which is also known as softmax function. Therefore, the output at k-th neuron can be calculated by following expression [12]:

The output of all the nine words can be calculated as:

$$y_k = Pr(\text{word}_k|\text{word}_{\text{context}}) = \frac{\exp(\text{activation(k)})}{\sum_{n=1}^{V} \exp(\text{activation(n)})}.$$

The output of all the nine words can be calculated as: 0.14307 0.09492 0.11444 **0.12416** 0.14928 0.12287 0.11943 0.14482 0.52485. The probability of the word "hole" is highlighted in bold while the error vector can be calculated by subtracting the probability vector from the target vector. The word2vec model learns the vector representation of the words by back-propagating the errors and updating the weights in the matrix WI and WO. In the CBOW model, the context is represented by multiple words for a given target word. For example, we would use "cat" and "mouse" as context words for the target word "hole". Therefore the input layer now needs modification, which would be WI matrix of both "cat" and "mouse" as the input to the hidden layer.

2.2. Skip-Gram

Taking the example of "cat" and "mouse" as context words and "hole" as the target word, the feature vector in the model would be $[000100000]^T$, while the outputs would have $[000010000]^T$and $[000100000]^T$as target feature vectors, respectively. In this example, instead of producing one vector of probabilities, two such vectors would be produced. The error vector for each output in this case can be calculated as described in the CBOW section. However, all error vectors are added up to adjust the weights via back propagation.

All the discussion on word2vec so far indicates that both CBOW and skip-gram models depend largely on word co-occurrence. Word co-occurrence describes the frequency of a word occurrence in a text corpus alongside other words. In the context of the network log data, most of the data repeats itself and almost all of the attacks have same data. For example, if we consider any attack or a normal data, only few of the features in the dataset change and all other features remain the same. Since the network data has a lot of word co-occurrence, word2vec should be perfect for use with this type of non-textual data extensively studied in this paper.

3. Our Methodology

Every machine learning algorithm first requires training with an input dataset. Similarly, our model first trains the word2vec model and converts network log data into vectors. After training the algorithm, our next step is to find the cosine distance from the generated word vectors. The inputs to the cosine distance tool are the different network attack types, e.g., smurf, neptune, pod, land, ipsweep, etc. The output will be words with an increasing order of similarity in terms of the cosine distance of the input word with respect to a particular output word.

The cosine distance results are then verified for their accuracy by understanding the nature of the attack followed by interpreting whether the output words are related to the input attack word or not. We train the dataset with both CBOW and Skip-gram models available in the word2vec algorithm. Our goal is to evaluate the two models and find the best suitable model for anomaly detection in network log data. The results obtained will form a basis of classification of the network log data, thereby allowing detecting the anomaly present in the network data.

An ultimate goal would be to build a real time detection system using the word2vec algorithm. After training on the network log data, the algorithm would provide information whether the activity in the network is normal or abnormal, i.e., if the network has been compromised.

3.1. Text to Vector Conversion

We evaluate word2vec on the MGHPCC (Massachusetts Green High Performance Computing Cluster). To study the algorithm, we obtain the word2vec toolkit from the Google archive [13]. After downloading the toolkit, we compile the source code to generate the executables and use the toolkit with the demo scripts provided. The word2vec toolkit provides various demo scripts, but we are particularly interested in using `demo-word.sh` as it serves our purpose of finding causes of anomalies in the network data. `demo-word.sh` requires a training file as an input. After training, the user can query, for example, `tcp`, and then it calculates the cosine similarity of different words from the train file and emits the output. The cosine distance closer to 1 means it is very similar to the entered word. On the other hand, if the distance is away from 1, i.e., closer to 0, that means it is not closely related to the entered word.

An example run to generate the word vectors is as follows: `./word2vec -train data.txt -output vec.txt -size 200 -window 5 -sample 1e-4`

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

TABLE 1. WORD2VEC PARAMETER DESCRIPTION.

Parameter	Description
-train <file>	Use text data from <file> to train the model
-output <file>	Use <file> to save the resulting word vectors/word clusters
-size <int>	Set size of word vectors; default is 100
-window <int>	Set max skip length between words; default is 5
-sample <float>	Set threshold for occurrence of words. Those that appear with higher frequency in the training data will be randomly down-sampled; default is 1e-3, useful range is (0, 1e-5)
-hs <int>	Use Hierarchical Softmax; default is 0 (not used)
-negative <int>	Number of negative examples; default is 5, common values are 3–10 (0 = not used)
-iter <int>	Run more training iterations (default 5)
-binary <int>	Save the resulting vectors in binary mode; default is 0 (off)
-cbow <int>	Use the continuous bag of words model; default is 1 (use 0 for skip-gram model)

`-negative 5 -hs 0 -binary 0 -cbow 1 -iter 3`. Details about the input parameters for the word2vec executable are described in Table 1.

3.2. Dataset

The word2vec algorithm is trained on a network log data obtained from the KDD Cup 1999 [6], [7]. We use this data as it is one of the most popularly evaluated public data for the network anomaly detection. The actual dataset is approximately 743 MB for training purpose, and we use 10% of the actual dataset (about 75MB) that contains 22 training attack types. An example row in the dataset is as shown: `0, tcp, http, SF, 334, 1684, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 9, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.33, 0, 0, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, normal.`

As shown above, each log comprises of 39 features, which can be divided into basic features, content features, and traffic features as described in [6], [7]. The details of individual feature in those 3 feature categories can be found in [6], [7], [14].

4. Evaluation

The first set of evaluation is divided into two parts: 1) The algorithm is trained without any pre-processing of the data, i.e., it is trained on raw data; and 2) The algorithm is trained with some pre-processing, specifically we normalize the data. We choose normalization as our pre-processing because most of the features have values between 0 and 1, but some other features, like the number of bytes send to source and destination, vary from 0 to tens of thousands. Normalizing feature values implicitly weights all features on a near equal footing. Standard normalization formula is applied on each column of the data set denoted as:

$$x' = \frac{x - min(x)}{max(x) - min(x)},$$

where x is original value and x' is normalized value.

TABLE 2. ATTACK TYPES FOR ENTIRE DATASET IN FOUR CATEGORIES.

Denial of service (DOS) (6)	Remote to Local (R2L) (8)	User to Remote (U2R) (4)	Probe (4)
Back	ftp_write	Buffer_overflow	Ipsweep
Land	guess_passwd	Loadmodule	Nmap
Neptune	imap	Perl	Portsweep
Pod	multihop	rootkit	satan
Smurf	phf		
teardrop	spy		
	warezclient		
	warezmaster		

As mentioned in the earlier section, the KDD Cup dataset has all 22 attack types, which can be furthered categorized into 4 main categories as follows [6], [7]:

- DOS: denial-of-service (e.g. neptune, smurf).
- R2L: unauthorized access from a remote machine (e.g. imap, multihop).
- U2R: unauthorized access to local super user (root) privileges (e.g., loadmodule, rootkit).
- probing: surveillance and other probing (e.g., ipsweep, portsweep).

Table 2 lists all 22 attacks divided into their respective category.

We trained word2vec to find only attacks that occur more than 5 times in the entire training data set because word2vec depends on word co-occurrence and we would need more training data to find proper semantics and syntax for these attacks. Therefore, 3 out of 19 attacks listed in Table 2 cannot be detected by the algorithm. We present our results in two parts: one using raw data and the other using normalized data. The sample output of the word2vec algorithm is given in Figure 2.

Word	Cosine distance
-0.016913342	0.894904
loadmodule	0.809164
imap	0.796738
0.849411551	0.790103
ftp_write	0.787436
1.295700132	0.773903
0.193104814	0.758390
-0.148174689	0.750930
12.65796616	0.735594
14.08203231	0.733957

Figure 2. Word cosine distance output from `demo-word.sh` when input "land" is given to the trained vocabulary.

4.1. Raw Training Data

The input parameters used for training the algorithm are kept same as the ones obtained from the demo scripts provided along with the word2vec toolkit. As word2vec provides two models, i.e., CBOW and skip-gram, and to get better understanding of the outputs obtained from which model better suits our data set, we train the data set with both the models first and compare their outputs.

Table 3 shows the attack type and its most relevant word along with cosine distance for both CBOW and Skip-gram.

978-1-5090-3525-0/16/$31.00 ©2016 IEEE

TABLE 3. THE MOST RELEVANT WORD OUTCOME WITH COSINE DISTANCE USING RAW TRAINING DATA.

Attack Type	CBOW		Skip-Gram	
	Word	Distance	Word	Distance
Back	4032	0.316248	8314	0.813475
Buffer_overflow	Loadmodule	0.744735	Loadmodule	0.741816
ftp_write	501760	0.654970	501760	0.798405
Guess_passwd	SH	0.519847	179	0.611706
Imap	RSTOS0	0.565767	RSTOS0	0.819486
Ipsweep	Nmap	0.479878	Eco_i	0.672210
Land	Buffer_overflow	0.438745	7971	0.690237
Loadmodule	Buffer_overflow	0.744735	Rootkit	0.778895
Multihop	5131424	0.613557	5049	0.820894
Neptune	0.07	0.739160	Ldap	0.653005
Nmap	SH	0.506103	SH	0.684762
Pod	1480	0.725592	1480	0.947083
Portsweep	Satan	0.657618	RSTR	0.732714
Rootkit	Buffer_overflow	0.597605	2425	0.827302
Satan	Portsweep	0.657618	Portsweep	0.615319
Smurf	511	0.822005	1032	0.812195
Teardrop	Satan	0.401689	Urh_i	0.591581
Warezclient	Teardrop	0.373490	2451	0.679937
Warezmaster	11376	0.789164	2191	0.755315

As a rule of thumb, we do not consider the outputs of the words having cosine distance less than 0.6 because cosine distance tells the similarity between two vectors [15]. That is, the cosine distance near 1 means that the output word is more closely related to the input word, and our analysis indicate that there is no meaningful outputs with cosine distance less than 0.6. Also, by setting the threshold to 0.60, we can reduce the search space for detecting anomaly. This eliminates many of the CBOW output words for the attack types like "back", "imap", "guess_passwd", "ipsweep", etc., and some of the skip-gram output words.

The output words in both CBOW and skip-gram with above 0.60 of cosine distance values can be summarized as:

- "back" gives 8314 as output word for skip-gram, but this number denotes number of bytes from destination to source and this is a very common feature of a DOS attack. Also, in the case of "ftp_write" and "guess_passwd", the output word is not present in the input attack type and also the output word is not related to it.
- Some of the attacks and their output words indicate some interesting intuition. For example, in case of "buffer_overflow", it shows output as "loadmodule", and for "loadmodule", it shows "rootkit" for skip-gram and "buffer_overflow" for CBOW, respectively. Also "portsweep" attack shows "satan" as the most relevant word and vice versa. All other attacks like "buffer_overflow", "loadmodule", "rootkit" fall into same category. For example, U2R and "portsweep", "satan" fall into same category, which confirms that they are all probing attacks.

While the above results provide some insight about certain attack types, it lacks intuition for other attacks. Even though the output word has a high cosine similarity with the input attack type, both the input attack and output words are irrelevant in many cases and do not depend on each other which can be observed by the description and signature of the attack [16].

Since this method did provide relatively less useful information on the attacks, we next apply normalization as a pre-processing on the data. Normalization scales the data by bring all the data on to near equal footing so that the algorithm can provide us with more useful results. We consider scaling or data normalization for pre-processing because it can be observed in the dataset that besides the features "number of data bytes from source to destination" and "number of data bytes from destination to source" all other features are in the range of 0 to 1, while the above two features vary from 0 to tens of thousands. Hence in the next section we discuss the results after training the algorithm on a normalized dataset.

4.2. Normalized Training Data

As discussed earlier, we have normalized the data and also have taken same approach as with the raw input data, i.e., only consider output words with cosine distance greater than 0.60. With the normalized data, we also obtain several outputs showing cosine distance greater than 0.60. Taking that into account, we have tabulated the results which can be found in [14], with five top most relevant words for the input attack types. Some of the attacks do not show cosine distance greater than 0.60 and hence we do not present in the table.

To better understand the results, we have categorized the output from word2vec into 4 distinct feature sets:

- **Exactly relevant features**: The input attacks which have the exactly matching relevant feature as output are grouped into this category.
- **Mixed results**: Input attacks having mixed features, i.e., either same category or some broad feature like output with attacks depending on general feature but not prominent one, are grouped into this category.
- **Irrelevant**: Output with no features matching to the input attacks are grouped into this category.
- **No Output**: Attacks which didn't give any output or whose cosine distance was less than 0.60 are in this category.

As shown in Figures 3, word2vec predicts more accurately with the normalized data as compared with the results without normalization. We also note that because we train with the normalized data, skip-gram method provides more meaningful results than the CBOW method; skip-gram identified related words for 16 attacks out of 19 compared to CBOW which identified only 11 attacks. We also make several additional observations:

- CBOW predicts root causes for certain attack types, but it fails for other attacks. For example, it did not find relevant words for back, imap, ipsweep and nmap, while skip-gram failed to find result only for nmap.
- Skip-gram predicts more words that have cosine similarity greater than 0.60 as compared to CBOW.

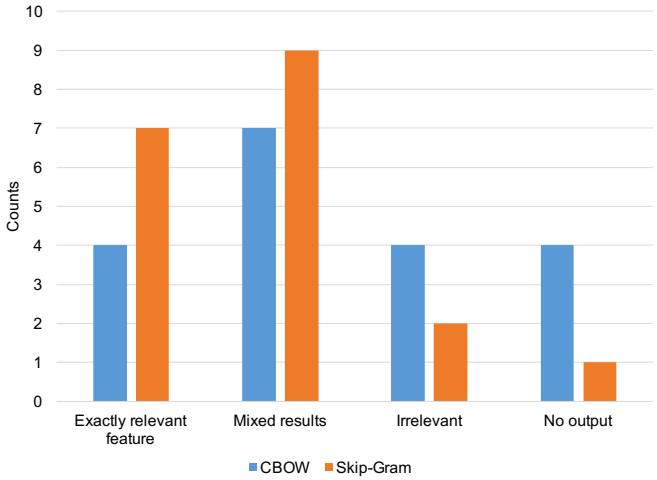

Figure 3. Results summary with CBOW and skip-gram model.

- Skip-gram predicts more meaningful results as compared to CBOW. Meaningful results as in the words predicted by skip-gram are more related to the particular attack type than the words predicted by CBOW. Source of some of the attacks like land, guess_passwd, pod, portsweep, smurf, neptune, teardrop were more clearly identified by skip-gram method than CBOW.

From all of the above points, it is clear that the skip-gram method with normalized training data provides more meaningful results. Also, the skip-gram method predicts results with much higher cosine similarity when compared to CBOW. Detailed description on each of the output for the various input attacks with both CBOW and Skip-gram method is given in [14].

4.3. Discussion

Convolutional Neural Network (CNN) Classifier. Word2vec proved to be useful to detect actual cause of attacks in the network log data, but it is not clear if we can use the word vectors obtained from word2vec to classify the data. The network log data can be classified in three ways: 1) binary classification which is classifying between normal data and anomalous data; 2) classifying data between normal data and 4 main attack categories (dos, u2r, r2l, probe); and 3) classifying each individual attack and the normal data.

To demonstrate feasibility of our study, we have used CNN model proposed by Yoon Kim [17] to classify the data using word vectors. We train a simple CNN with one layer of convolution on top of word vectors obtained from word2vec. The CNN model used only classifies between positive and negative data, i.e., binary classification of the data. The overall validation performance on our network log data is about 99.5%. We speculate this is an overfit because no algorithm provides 100% accuracy for validation and training. Even if we obtain an overfit performance with CNN, the main point to prove here is that we can still use word vectors with deep neural network algorithms like CNN.

Vector Quantization. One possible reason we were able to detect only 7 attacks out of 19 with exactly matching feature could be word occurrence. It is clear from the results that, if we increase the word occurrence, then we may obtain more accurate results for finding the most relevant feature for a certain attack. One way to achieve this is to apply a vector quantization technique because it provides a way to divide a large set of vectors into groups having approximately the same number of vectors closest to them. For example, if there are 2 vectors 1.01234 and 1.01222 in a feature set, word2vec would consider these 2 vectors as unique words but it may be the case that both vectors do not provide any significant difference to classify between normal and abnormal attack. Vector quantization provides functionality to combine these near vectors into a single vector which represents both of these vectors. This would in turn increase the frequency of word occurrence, leading us to achieve better results.

Both the above discussions prove to be helpful for future works on this study, which can largely increase the accuracy of results from word2vec. Our motivation was to implement a natural language processing algorithm on non-textual data which we prove to be feasible and this additional research and discussion increases the scope of this study.

5. Conclusion and Future Work

In conclusion, we demonstrate that Natural Language Processing can not only be used for textual data, but also can be similarly implemented for non-textual data with some preprocessing. We further show that at least 7 out of the given 21 attacks were detected with exact matching features for that particular attack. Lastly, we are able to successfully identify 85% of all of the factors for a particular attack. Our study proves our motivation behind this study. As a text data comprising of words and sentences have a grammatical structure in the same way non-textual data has a underlying structure, i.e., both have same type of structure and hence non-textual data also work with NLP techniques just as text data works.

In our future work, we intend to exploit the word2vec extensively by tweaking several arguments in the word2vec script, which include: size, window, min-count, etc. We plan to apply different machine learning-based classification algorithms like decision tree or random forest. We also plan to investigate more into the CNN classifier that we discussed in earlier section in order to improve classification performance. While not thoroughly applied in our evaluation, we expect vector quantization would be promising to improve our results. In our future work, we would like to explore this technique extensively and to see how it effects the word2vec output.

References

[1] "PandaLabs' Annual Report 2015," Panda Security, Tech. Rep., 2015. [Online]. Available: http://www.pandasecurity.com/mediacenter/src/uploads/2014/07/Pandalabs-2015-anual-EN.pdf

[2] Wikipedia, "Anomaly detection." [Online]. Available: https://en.wikipedia.org/wiki/Anomaly_detection

[3] L. Mechtri, F. D. Tolba, and N. Ghoualmi, "Intrusion detection using principal component analysis," in *Second International Conference on Engineering Systems Management and Its Applications (ICESMA)*, March 2010, pp. 1–6.

[4] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, 2001, pp. 5–8.

[5] "Bag of words meets bags of popcorn." [Online]. Available: https://www.kaggle.com/c/word2vec-nlp-tutorial

[6] "KDD Cup 199 Data." [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[7] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA Off-line Intrusion Detection Evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, Oct. 2000.

[8] "word2vec." [Online]. Available: https://www.kaggle.com/c/word2vec-nlp-tutorial

[9] "What are the continuous bag of words and skip-gram architectures, in layman's terms?" [Online]. Available: https://www.quora.com/What-are-the-continuous-bag-of-words-and-skip-gram-architectures-in-laymans-terms

[10] M. Caudill, "Neural networks primer, part i," *AI Expert*, vol. 2, no. 12, pp. 46–52, Dec. 1987. [Online]. Available: http://dl.acm.org/citation.cfm?id=38292.38295

[11] "A basic introduction to neural networks." [Online]. Available: http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[12] "Continuous Bags of Words (CBOW)." [Online]. Available: https://iksinc.wordpress.com/tag/continuous-bag-of-words-cbow/

[13] "word2vec." [Online]. Available: https://code.google.com/archive/p/word2vec/

[14] K. Barot, "Using Natural Language Processing on Non-Textual Data: A Case Study of Network Anomaly Detection," Master's thesis, University of Massachusetts Lowell, 2016.

[15] [Online]. Available: https://en.wikipedia.org/wiki/Cosine_similarity

[16] "DARPA Intrusion Detection Evaluation." [Online]. Available: https://www.ll.mit.edu/ideval/

[17] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.