Extracting Tactics from Cybersecurity Articles

Jake Smola
smola@stanford.edu

I. Introduction

Emerging cybersecurity threats can pose significant risk to organizations and their customers, often resulting in the compromise of confidential data and other devastating effects with substantial legal and business costs. By maintaining awareness of the modern threat landscape, at-risk organizations can evolve their defensive measures at the pace of the adversary and prevent successful attacks. One way such organizations accomplish this is by consuming research articles and extracting the attackers' tactics from each document. Doing so allows these companies to prioritize their threat detection and prevention capabilities. However, this task is time and labor intensive and not all organizations are equipped to handle it successfully. We seek to address this problem by training an automated text classifier which can predict the attack tactics described in a given document. Specifically, our solution takes as input the text from a particular document and outputs a subset of up to seven attack tactics that the document describes, as illustrated in figure 1. The multi-label classifier is part of *Magpie*, an open-source text classification tool used by the European Organization for Nuclear Research (CERN). *Magpie* leverages the word2vec and convolutional neural network models to train on and subsequently classify text samples. We will describe these models and the overall classification scheme in greater detail in section IV. In the next section, we will describe some complementary text classification models.

II. Related Work

Multi-label text classification can be broken down further into two distinct steps: feature extraction and classification. Text must first be reduced to a computable format (feature extraction) before a classification model can be trained to make predictions on new data.

Several models for text feature extraction exist, and their implementation generally coincides with word embedding, where word from a pre-selected vocabulary are mapped to vectors of real numbers. Among the most popular such models is *Term Frequency Inverse Document Frequency* (TF-IDF) [13]. TF-IDF quantifies the importance of a term in a given document and is computed by multiplying the normalized term frequency of a given vocabulary term by the inverse of the proportion of documents that contain the term. One can compute the TF-IDF of each vocabulary term over any document to obtain a vector of TF-IDF values. TF-IDF takes little time to compute but is not as good as other methods in considering local context within the corpus. A more modern

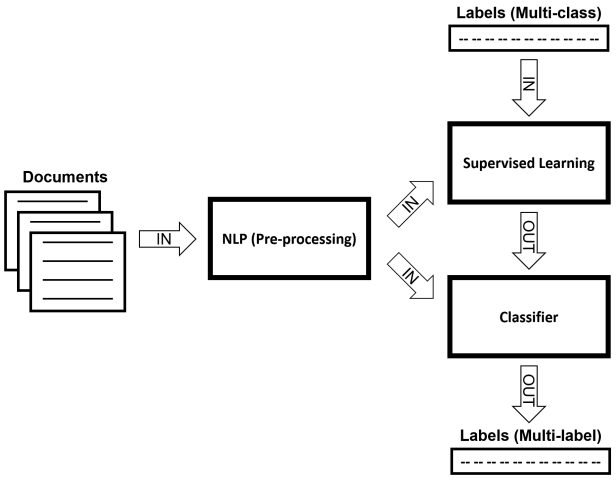

Fig. 1. Multi-label classification model.

alternative to TF-IDF is termed *Word2Vec* [10]. The *Word2Vec* algorithm takes as input a corpus (multiple documents worth of text), a window size, and an output dimensionality and uses a continuous bag-of-words (CBOW) or skip-gram model to learn the word vector corresponding to each word in the corpus. We will describe *Word2Vec* in further detail in section IV. *Word2vec* is a powerful model that does consider local context, unlike TF-IDF. However, the underlying model can treat unique permutations of the same set of words (e.g. sentences with the same words but different word order) equally, in addition to neglecting word semantics [8]. An extension of *Word2Vec* called *Doc2Vec*, seeks to correct for this potential issue [8]. *Doc2Vec* builds upon the model of *Word2Vec* with the addition of paragraph vectors that are derived or inferred using an architecture similar to that of *Word2Vec*. Although its authors claim *Doc2Vec* is able to correct for some of *Word2Vec*'s aforementioned problems, an insightful comparison of the above feature extractors shows the two models can perform roughly equally on a large dataset [2].

Like feature extraction, the classification portion of multi-label text classification has been studied for some time. We note that we only consider classification in the supervised case. Both discriminative and generative models exist, but most are generalizations of single-label models or unsupervised models. [15] and [6] show a variety of these transformations

for discriminative models such as SVM, k-Nearest Neighbors, adaBoost, logistic regression, and C4.5. Many such transformations are made using the binary relevance method or one-vs-all. In terms of generative models, [18] provides a comparable transformation of the Naive Bayes classifier to the multi-label setting. Labeled LDA [12] and Supervised LDA [17] adapt the unsupervised *Latent Dirichlet Allocation* (LDA) [3] to support supervised learning in the multi-label setting.

Implementations of multi-label text classification tend to utilize combinations of the aforementioned models. The recently published *DocTag2Vec* leverages extends *Doc2Vec* with a multi-label k-Nearest Neighbors implementation to facilitate label prediction [5]. *Magpie*, a text classification product built by the European Organization for Nuclear Research (CERN), employs *word2vec* for feature extraction and multi-label logistic regression to classify text over a convolutional neural network [14].

III. Data and Features

In this section, we describe in detail the data we utilized in our experiments, the label convention we train our model to predict, pre-processing associated with the data, and the features extracted from our data.

A. Data

The data we leverage in our experiments consists of Palo Alto Networks Unit 42 blog posts [16]. These blog posts are ideal for our task as they tend to be focused on cyber attack TTPs. As of the time of this writing, there are over 300 Unit 42 blog posts on record.

B. Labels

For this project, we have chosen to use a multi-label convention consisting of labels of the following format:

$$\left\{\begin{array}{c}\text{Discovery}\\ \text{Privilege Escalation/Credential Access}\\ \text{Execution}\\ \text{Lateral Movement}\\ \text{Command \& Control}\\ \text{Exfiltration/Collection}\\ \text{Persistence/Defense Evasion}\end{array}\right\} \in \{0,1\}^7$$

Inherently, these labels represent a consolidated sequence of tactics considered by Palo Alto Networks and [1]. In most implementations, these labels will be represented by a single vector of 1's and 0's, where each element represents the discussion (1) or lack-thereof (0) of a particular tactic.

Unfortunately, our dataset is not labeled, and we had to label the data manually. Due to time constraints, we have only labeled 50 of the blog posts in preparation for our experiments.

C. Data Pre-processing

To prepare the blog posts for interpretation by *Magpie*, we first scraped the text body of each post available from [16] prior to converting the post to raw ascii format. Figure 2 displays an excerpt from one of our scraped data samples.

```
The Trojan uses the first four bytes of
this hardware ID as a unique identifier
for the system, which in our case was
0000. Figure 3 Hardware ID used by XAgent
to uniquely identify compromised hosts
When generating the URLs within the
HTTP POST and GET requests, XAgent sets
one HTTP parameter using a specific
data structure that contains this
agent_id value. This parameter transmits
the agent_id to the C2 server to
obtain commands the actor wishes to
execute on the compromised system. The
data structure used to transmit the
agent_id to the C2 is as follows: struct
DWORD random_value_for_key; CHAR[7]
constant_value; DWORD agent_id;  The
constant value in the data structure
is a 7 byte string that is hardcoded to
\x56\x0E\x9F\xF0\xEB\x98\x43, followed by
the agent_id value (0000 in our case).
```

Fig. 2. Excerpt from a Palo Alto Networks Unit 42 Blog Post.

In preparation for our experiments, we sampled three random permutations of the 50 labeled blog posts and fixed the five final posts from each permutation as the test set for that permutation.

Additional NLP pre-processing tasks, including tokenization, stemming, special character stripping, and case conversion, are handled by *Magpie*.

D. Features

After the remaining pre-processing tasks are handled by *Magpie*, it passes the training corpus to the *Word2Vec* model for feature extraction. *Word2Vec* maps each word in the entire corpus to a vector living in n-dimensional space, where n is the dimensionality parameter passed to the model. In section V we describe how we vary this parameter during the course of our experiments.

IV. Methods

In this section we highlight the two primary models utilized by *Magpie*: *Word2Vec* and a convolutional neural network.

A. Word2Vec

Word2Vec constitutes the second stage of Magpie's training pipeline after data pre-processing. The algorithm takes as input a corpus (multiple documents worth of text), a window size, and an output dimensionality n. The implementation of Word2Vec in Magpie comes from the gensim library and leverages the continuous bag-of-words (CBOW) model (default) to learn a the word vector corresponding to each word in the corpus [11].

Word2Vec first iterates over the corpus to generate the *vocabulary* of size V which consists of a mapping from each

encountered word i to its one-hot representation, the vector x_i. This representation is then passed to the CBOW architecture. At a high level, CBOW seeks to predict the current word given the surrounding words or *context* [9], where the size of the context C, is determined by the window size parameter. The model randomly initializes two matrices, W and W', of dimensions $V \times n$ and $n \times V$ respectively [4]. During forward propagation, each one-hot input from the specified context is then fed through the neural network as defined in figure 3. Note that the output layer is the only layer with an activation function, computing the softmax on the preceding score to produce the predicted target word $\hat{y}$. During back propagation, the model seeks to minimize the loss defined by:

$$J = -p(W'_z|h) = -W'^T_z \cdot h + \log \sum_{i=1}^{V} W'^T_i \cdot h$$

where z is the index of the target word in W'.

Word2Vec does this using stochastic gradient descent. When optimized, W' contains all word vectors which are then passed to the next stage in the pipeline for classification using a convolutional neural network.

B. Convolutional Neural Network

A convolutional neural network (CNN) is a specialized neural network which can make accurate predictions in a highly dimensional space without necessarily using a high volume of memory. In order to reduce the number of parameters that must be computed, CNN's apply the same weights to local regions of the original input rather than computing weights for each region independently.

CNN's are typically associated with a variety of hyperparameters, including:

- Receptive field: the fixed size of each local region being considered.
- Depth: the desired output vector length, otherwise known as the number of "filters" used.
- Stride: the number of units by which the receptive field is shifted among consecutive local regions.

There are several other hyperparameters for a typical CNN, however, we limit our discussion to the above three for clarity and succinctness.

Figure 4 depicts a primitive convolutional neural network with a depth and stride of 1 and a receptive field of 3, where the input layer is represented by circles and the first hidden layer is represented by squares. H represents the homogenous score and activation calculations that happen for each local region of input.

During training, *Magpie* invokes a convolutional neural network with a depth of 256 and stride of 1 and pools results for receptive windows of 1-5. The input is a three dimensional array mapping documents to word vectors. For forward propagation, the first hidden layer (called the convolution layer) uses tanh as its activation function, while the output layer uses the sigmoid function. For back propagation, the model optimizes

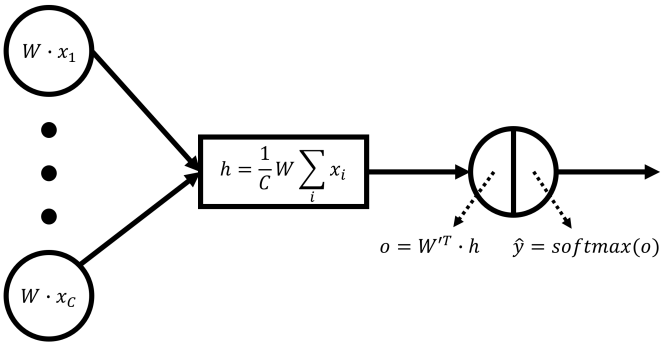

Fig. 3. Continuous bag-of-words architecture with the following layers from left to right: input, hidden, output.

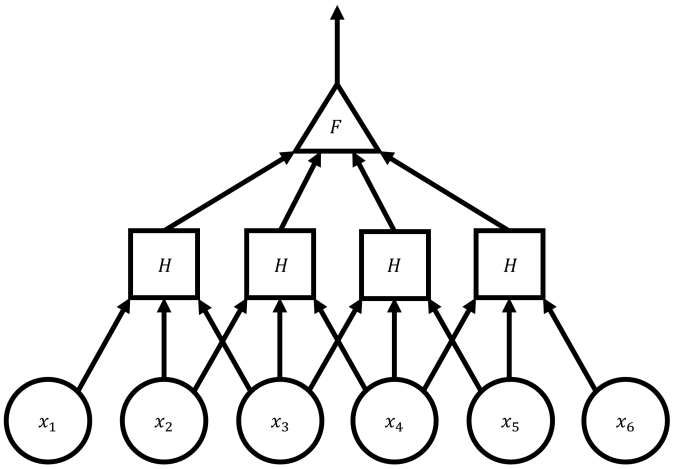

Fig. 4. Convolutional neural network with a depth and stride of 1 and receptive field of 3 where circles represent the input layer and remaining nodes represent hidden layers.

binary cross-entropy loss using the Adam algorithm. We spare the details of these methods as they are interchangeable with other, more common methods.

To make a prediction on a given document, *Magpie* conducts the usual pre-processing and then creates a matrix containing word vectors for each word. Words that are not in the trained *Word2Vec* model are assigned a vector of zeros. Forward propagation then outputs a 256-coordinate vector representing the probabilities of each label given the input document. The labels are sorted by their probability and the highest ranking labels are returned.

V. Results

We hypothesized that the limited size of our labeled dataset would inhibit optimal classification performance. Furthermore, we recognized that our problem concerned a variety of hyperparameters that would have to be tuned regardless of the size of our dataset. Therefore, we decided to focus our work on (1) validating whether or not *Magpie* does indeed converge to an acceptable accuracy (we aim for 90% test accuracy given the current labeled dataset size); (2) exploring the effects of a subset of hyperparameters in hopes that future extensions of

this work could direct resources to other parameters or aspects of the problem. We varied the below hyperparameters to gauge their effects:

1) *Magpie*'s validation ratio: This parameter specifies the proportion of provided data samples that will be withheld from training and reserved for validation after each CNN epoch. Throughout our experimental process, the test set consisted of five samples while the remaining 45 samples were impacted by the choice of validation ratio.
2) Feature set dimensions: The number of dimensions associated with the feature space to which all words in the corpus are mapped by *Word2Vec*.
3) Number of CNN epochs: the number of training iterations conducted during each instance of classification training.

We conducted the following experiments:

1) Experiment 1: Coarse exploration of the effects of *Magpie*'s validation ratio, feature set dimensions, fixing the number of CNN epochs at the default value of 100.
2) Experiment 2: Coarse, informed exploration of the effect of the number of CNN epochs, reducing feature set dimensions to a subrange of those in Experiment 1 and fixing the validation ratio at 10% based on Experiment 1 results.

In both experiments, for every chosen combination of parameters, we trained the classifier three times—once for each of the three data permutations—and computed the average training and test error. We rely on these metrics for our analysis.

We will discuss the results for each experiment in turn.

A. Experiment 1

In experiment 1, we considered validation ratios of 10% and 20%, as well as feature set dimensions of 100, 200, 300, 400, and 500. The CNN was trained over 100 epochs. Figures 5 and 6 report the average training and test error from experiment 1.

To our surprise experiment 1 reveals training and test errors that are rather acceptable for an untuned, multi-label text classifier with few training samples. We noticed an average training error as low as 4.5% and a test error as low as 12.5%. Given the stability of loss over the last several epochs, this result tells us that without significant tuning, *Magpie* converges comes within 2.5% of our desired outcome of 90% test accuracy and continued experimentation is thus warranted.

In regards to the effects of validation ratio, we see that this ratio affects error systematically and consistently for both training and testing. Training with a validation ratio of 0.2, which corresponds to a 36-9-5 training-validation test split, leads to consistently higher error than training with a validation ratio of 0.1, which corresponds to a 40-5-5 split. Because documents in the validation set do not impact training directly, this result confirms our hypothesis: that the limited size of our dataset will hurt classification performance. It is evident that increasing the number of training samples will likely continue to improve our performance with diminishing returns.

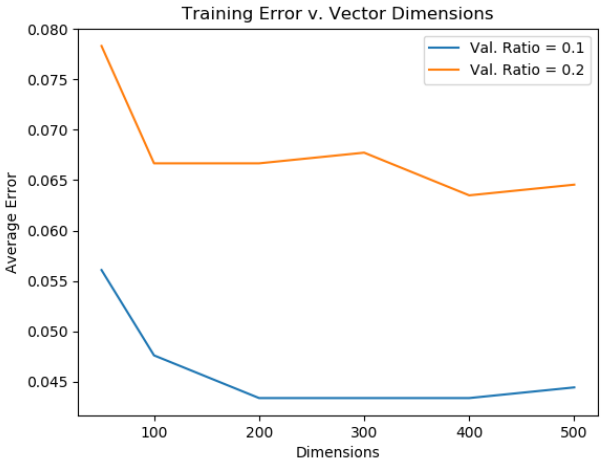

Fig. 5. Experiment 1: Average Training Error

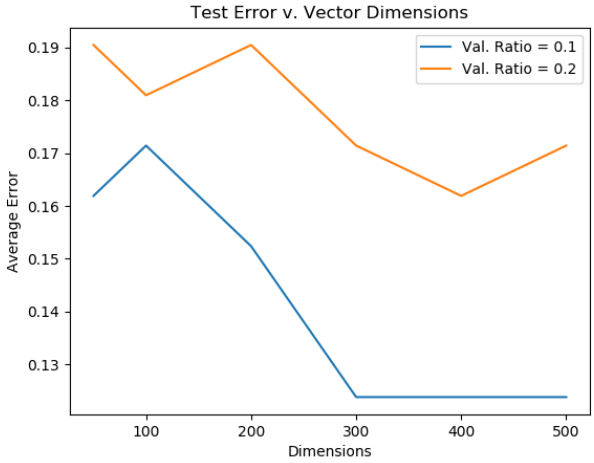

Fig. 6. Experiment 1: Average Test Error

In regards to our feature set dimensions, we see that the dimensionality of the feature set has a higher impact on test error than training error. We also see that as the classifier appears to achieve a global minimum average error (given the available values for our hyperparameters) when using a feature set between 300 and 400 dimensions. This indicates that the optimal choice for feature set dimensionality may reside on this subrange and we thus chose re-assess a narrower range of feature-dimension classifiers in experiment 2.

B. Experiment 2

Given the findings of experiment 1, in experiment 2 we fixed the validation ratio at 10% and reduced the considered feature set dimensions to only 200, 300, and 400. To test the impact of the number of CNN epochs, we increased the number of epochs to 150. The results of experiment 1 could therefore

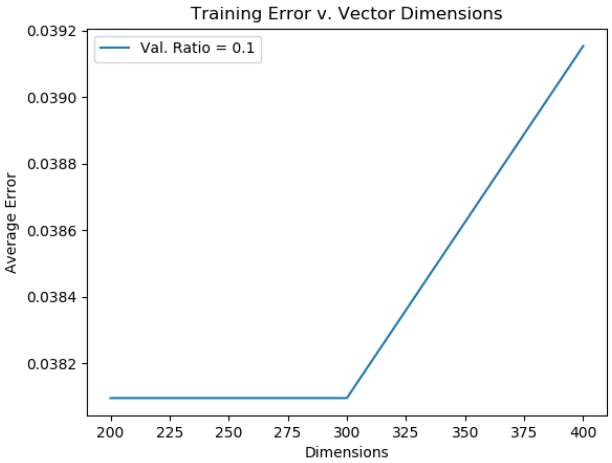

Fig. 7. Experiment 2: Average Training Error

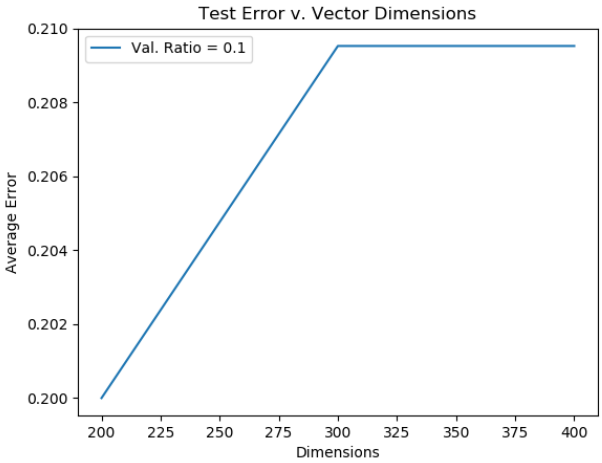

Fig. 8. Experiment 2: Average Test Error

be used as a baseline for comparison against experiment 2. Figures 7 and 8 report the average training and test error from experiment 2.

In experiment 2 we saw a relative decrease of approximately 0.5% in training error across all feature set dimensions covered in both experiments. This shows that additional epochs improve training accuracy and thus the model is indeed converging.

Additionally, all three tested dimensions saw an average training error within a window of 0.11%, which is consistent with experiment 1. This leads us to believe that the effects of the number of epochs and the number of feature set dimensions are largely orthogonal and can be considered in isolation.

In regards to test error, experiment 2 incurred significant higher test error across all feature set dimensions covered in both experiments; in some cases this error was nearly twice that in experiment 1 for comparable trials. Since the training error has improved but the test error has decreased, it is evident that the additional epochs have induced over-fitting during training. The ideal number of epochs thus likely sits below 150 and future trials should complete below this bound given the dataset and the choice of model arguments. This comes as no surprise given the small dataset used in training.

VI. Conclusion

During our experiments, we confirmed that our current labeled dataset is insufficient for our needs and additional labeling must be done to improve the test accuracy of our model and reduce over-fitting. Additional hyperparameter tuning can also improve test accuracy. However, we also learned that even with an undersized dataset and limited tuning, *Magpie* converges and can perform fairly well in classifying documents in the multi-label setting; it is thus a great candidate for use in future experiments of this nature.

In follow-on experiments, we wish to incorporate more labeled data and additional hyperparameter tuning to improve the model's accuracy on our dataset. Additionally, we hope to modify *Magpie* to employ regularization in the back propagation phase of the CNN alongside early stopping when consecutive epoch results do not appear to improve. Other interesting variations to our work include the use of an existing *Word2Vec* model instead of a model computed from the training documents and employing *Magpie*'s recurrent neural network instead of the default CNN to compare their relative performance.

VII. Contributions

I would like to thank Prajakta Jagdale of Palo Alto Networks for labeling half of the 50 training samples used in our analysis. Without these labels, our results likely would have suffered from even higher variance and error. I would also like to thank Poonam Bhide, a former CS229 peer who helped with drafting the initial project proposal and shaping the project's higher-level direction.

References

[1] Adversarial Tactics, Techniques & Common Knowledge. *MITRE*. Available from: https://attack.mitre.org/wiki/Main_Page

[2] Batista, D. S. Document Classification. Available from: http://www.davidsbatista.net/blog/2017/04/01/document_classification/

[3] Blei, D., Ng, A., and Jordan, M. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993-1022, 2003. Available from: http://ai.stanford.edu/~ang/papers/nips01-lda.pdf

[4] Chaubard, F., Mundra, R., Socher, R. CS224d: Deep Learning for NLP: Lecture Notes Part I. Available from: https://cs224d.stanford.edu/lecture_notes/notes1.pdf

[5] Chen, S., Soni, A., Pappu, A., and Mehdad, Y. DocTag2Vec: An Embedding Based Multi-label Learning Approach for Document Tagging. Available from: https://arxiv.org/abs/1707.04596

[6] Cheng, W., Hllermeier, E. Combining instance-based learning and logistic regression for multilabel classification. Machine Learning, 76(2), pp.211-225, 2009.

[7] Convolutional Neural Networks (CNNs/ConvNets). Available from: http://cs231n.github.io/convolutional-networks/#layers

[8] Le, Q.V., Mikolov, T. Distributed representations of sentences and documents. In Proceedings of the 31 st International Conference on MachinevLearning, Beijing, China, 2014.

[9] Mikolov, T., Chen, K., Corrado, G., Dean, J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013. Available from: https://arxiv.org/abs/1301.3781

[10] Mikolov, T. Sutskever, I., Chen, K., Corrado, G., Dean, J. Distributed Representations of Words and Phrases and their Compositionality. Computation and Language. Available from: https://arxiv.org/abs/1310.4546
[11] Models.word2vec Deep learning with word2vec. Gensim. Available from: https://radimrehurek.com/gensim/models/word2vec.html
[12] Ramage, D., Hall, D., Nallapati, R. and Manning, C.D. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, 1, pp. 248-256, 2009.
[13] Salton, G. and McGill, M.J. Introduction to Modern Information Retrieval. McGraw-Hill Book Co., New York, 1983.
[14] Stypka, J. *Magpie*. *Github*. Available from: https://github.com/inspirehep/magpie
[15] Tsoumakas, G., Katakis, I. Multi-label classification: An overview. International Journal of Data Warehousing and Mining, 3(3), 2006.
[16] Unit 42. *Palo Alto Networks*. Available from: https://researchcenter.paloaltonetworks.com/unit42/
[17] Wang, C. Supervised latent Dirichlet allocation for classification. Available from: http://www.cs.cmu.edu/~chongw/slda/
[18] Wei, Z., Zhang, H., Zhang, Z., Li, W., Miao, D. A naive Bayesian multi-label classification algorithm with application to visualize text Search Results. International Journal of Advanced Intelligence, 3(2), pp.173-188, 2011.