On using pretext tasks to learn representations from network logs

On Using Pretext Tasks to Learn Representations from Network Logs

Matteo Boffa, Luca Vassio, Marco Mellia
Politecnico di Torino
Italy
first.last@polito.it

Idilio Drago
Università di Torino
Italy
idilio.drago@unito.it

Giulia Milan, Zied Ben Houidi, Dario Rossi
Huawei Technologies Co. Ltd
France
first.(mid.)last@huawei.com

ABSTRACT

Learning meaningful representations from network data is critical to ease the adoption of AI as a cornerstone to process network logs. Since a large portion of such data is textual, Natural Language Processing (NLP) appears as an obvious candidate to learn their representations. Indeed, the literature proposes impressive applications of NLP applied to textual network data. However, in the absence of labels, objectively evaluating the goodness of the learned representations is still an open problem. We call for a systematic adoption of domain-specific pretext tasks to select the best representation from network data. Relying on such tasks enables us to evaluate different representations on side machine learning problems and, ultimately, unveiling the best candidate representations for the more interesting downstream tasks for which labels are scarce or unavailable.

We apply pretext tasks in the analysis of logs collected from SSH honeypots. Here, a cumbersome downstream task is to cluster events that exhibit a similar attack pattern. We propose the following pipeline: first, we represent the input data using a classic NLP-based approach. Then, we design pretext tasks to objectively evaluate the representation goodness and to select the best one. Finally, we use the best representation to solve the unsupervised task, which uncovers interesting behaviours and attack patterns. All in all, our proposal can be generalized to other text-based network logs beyond honeypots.

CCS CONCEPTS

• **Networks** → **Network security**; • **Computing methodologies** → **Knowledge representation and reasoning**.

KEYWORDS

Representation Learning, Cybersecurity, NLP, Honeypots

ACM Reference Format:
Matteo Boffa, Luca Vassio, Marco Mellia, Idilio Drago, and Giulia Milan, Zied Ben Houidi, Dario Rossi. 2022. On Using Pretext Tasks to Learn Representations from Network Logs. In *Native Network Intelligence (NativeNI '22), December 9, 2022, Roma, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3565009.3569522

NativeNI '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9887-9/22/12…$15.00
https://doi.org/10.1145/3565009.3569522

1 INTRODUCTION

The success of deep learning in areas such as computer vision (CV) and natural language processing (NLP) partly comes from the relatively standard input format from which specialized models can learn solid representations that are useful across different tasks. As the success of AI spreads, the networking community is considering options to natively integrate it in networks, as a way to ease its adoption and take full profit from its potential. Learning how to best represent network data is the basis for pursuing such objectives.

However, there is no consensus on what is the equivalent of an image or a document for network data. The latter is complex and multi-modal by nature: it may come from flow monitors, server logs, security systems, etc. Also, information changes rapidly over time (e.g., the birth of new services, failures, attacks), space (e.g., new servers or CDN nodes, new routes, new botnets) and semantically (e.g., new protocols and new vulnerabilities).

Fortunately, despite this heterogeneity, a large portion of network data is textual, putting NLP techniques in a good position to learn representations from it. Noticing this similarity, recent work successfully leveraged NLP techniques to learn representations from non-natural languages, including in networks. Examples include programming languages (e.g., code2vec [5], CodBert [13]), configuration languages [8] and also categorical data such as ports and IP addresses [14, 21] for which word embeddings [16] were used to learn representations. Following this trend, recent work [6, 9, 12, 22, 24] showed promising results on applying NLP techniques also on bash logs for network security purposes.

Unlike other domains, however, network data labels are often scarce, leaving few-shot and unsupervised learning as the only options. Luckily, if sufficiently good representations – or embeddings – are learned from complex data, then few labels are often enough to solve the desired downstream task. This was recently demonstrated for both computer vision [23] and natural language processing [10]. In these cases, unsupervised or self-supervised techniques can be used to learn representations. A prominent example is word2vec [16] (W2V), which we also use here. It assigns vectors to words by learning to predict a word from it context. However, a big challenge is how to know if the unsupervisedly learned representations are of good quality. Indeed, different W2V models and different hyper-parameters leads to different embeddings. How to select the best is thus critical.

In this paper we propose to systematically use objective domain-specific *pretext tasks* to measure the quality of intermediate representations of network data. In machine learning, pretext tasks are side tasks that are not necessarily useful, but for which labels are cheaply available. Historical examples include hiding a portion of

an image and learning to impute it (in CV), or learning to predict the next word or sentence in a given context (in NLP). Intuitively, the better the model performs on such tasks, the better the intermediate learned representations are. When exploring different models and different hyper-parameters, we thus select the one that yields the best results on the pretext tasks.

For our use case, we focus on network security and more specifically on honeypot SSH logs. Honeypots are efficient sensors to collect data about ongoing attacks. However, the data they expose is complex by nature, given the unknown and stealthy goals of attackers [18]. Successfully learning meaningful representations from such logs can drastically ease the work of security analysts even in the absence of labels. For example, it is useful to automatically group together (unknown) events that belong to the same threat and detect novel threats that were never seen before.

Having clustering as a downstream task in mind, we apply the following pipeline on honeypot logs. We generate different word embedding models to obtain a self-learned representation for tokens present in the logs. Given a set of representations obtained with different parameters, we design pretext tasks in the form of objective classification tasks, and use their performance as a proxy to choose the best representations. We next apply clustering on the best representations, qualitatively observing that they capture useful information.

2 PRETRAINING PIPELINE

2.1 Dataset

We rely on data collected using Cowrie [19], a honeypot that emulates a UNIX shell exposed via Telnet and SSH. Cowrie allows attackers to access the emulated server and saves all attackers' shell inputs. We call an *attacking session* the inputs captured after the login until the attacker's logout. As part of its engagement strategy, Cowrie provides attackers with a virtual file system, generates realistic outputs for hundreds of inputs, and downloads files and scripts as requested by the attackers. Cowrie record all sessions in textual logs, serving as a valuable source of information for security analysts. Sessions in logs are however difficult to analyze given the unpredictable behaviour of attackers.

Here, we use open data from the Honeypot as a Service (HaaS) project [3], which aggregates logs of attacks observed by a distributed network of volunteers' devices. All attempts are proxied to Cowrie honeypots in the cloud. We consider ~ 175 000 sessions in total, and use the dataset both in our pretext tasks and on a downstream ML task. Clearly, sessions have no labels that we can leverage.

2.2 Representation learning and Word2Vec

Representation Learning plays a crucial role in domains such as NLP and CV. As we focus on textual network logs, we restrict ourselves to representations learned from sequences of textual tokens. In particular, we investigate the quality of different representations learned with W2V, one of most famous and useful embedding tool in NLP.

W2V exploits a simple neural architecture to leverage word co-occurrences in a sentence and project them into a high-dimensional space, i.e., the embedding space. An embedding is a numerical representation for a given word. The implicit goal of W2V is to project "similar" words (i.e., words that generally appear in similar contexts) into neighboring regions of the embedding space.

Table 1: W2V parameters. Each combination generates a different embedding.

Parameter	Preliminary Study	Deeper analysis
W2V model	["CBOW", "Skip Gram"]	["Skip Gram"]
Context window [w]	[3, 5, 10]	[2, 4, 6, 8, 10, 12, 14]
Negative samples [ns]	[2, 5, 10]	[2, 4, 6, 8, 10, 12, 14]
Vector size [H]	[100]	[20, 50, 80, 100, 150, 200, 300]
total W2V models	18	343

The algorithm has been applied to scenarios outside NLP, such as to learn embeddings for programming languages [5]. We here face a similar scenario, in which network logs are our documents, containing words and sentences.

W2V receives as input a set of N sentences S, that we use as *training corpus*. W2V scope is to learn, for each token c, an embedding $C \in \mathbb{R}^H$, where H determines the size of the embedding, an hyper-parameter of the model. W2V can have two objectives, known respectively as *Skip-Gram* and *CBOW* [15]. Consider the i-th position of a token in the sentence j, c_{ij}: Skip-Gram guesses the w tokens around c_{ij} $[c_{(i-\frac{w}{2})j}; c_{(i+\frac{w}{2})j}]$, where w is an hyper-parameter defining the *context window*. Contrarily, CBOW receives as input the w tokens around c_{ij} $[c_{(i-\frac{w}{2})j}; c_{(i+\frac{w}{2})j}]$ and has to find c_{ij}. The *negative samples* parameter *ns* [17] represents the number of negative examples of c_{ij}, i.e., tokens not present in the word context window. The idea is that, while W2V wants to maximize the closeness of c_{ij} with actual tokens in its context window (neighbors), it also wants to maximize the distance to other tokens (non-neighbors). Choosing the right number *ns* of negative samples is an open problem [7, 11].

Each combination of parameters generates a different embedding. Choosing which one results the best is complicated. Here we consider all models generated by any combinations of parameters in Table 1.

2.3 Pretext tasks

As there is no general way to measure the quality of W2V embeddings, we propose to leverage pretext tasks to evaluate the learned representations. We design two tasks as follows.

2.3.1 Token type classification. The goal of our first pretext task is the classification of Bash token types. We consider each session to be a *document* and each input line to be a *sentence*. Then, we parse all inputs with *Bashlex* [2] to split sentences into tokens, thus generating our training corpus (i.e., ingress words for W2V).

Bashlex let us also identify the token types, i.e., whether each token refers to commands, flags, parameters or separators. We benefit from those easy-to-obtain labels to construct our pretext task. In details, tokens belong to three categories, that we use to train supervised classifiers: (i) *commands*, which includes both builtin bash commands and executable files, (ii) *parameters*, (iii) *flags*, and (iv) *operators*.

For example, given the line received from an attacker

```
curl -O http://10.0.0.1/file; ./file | chpasswd;
```

we identify three commands (`curl`, `./file` and `chpasswd`), one

parameter (`http://10.0.0.1/file`), one flag (`-O`), and two operators (`|` and `;`).

Overall, our dataset contains 3 162 commands, 158 838 parameters, 462 flags, and 6 separators. Since the separators represents a minute fraction of the total we chose to exclude them from the classification task. The pretext task therefore consist in classifying tokens into 3 classes, using as input features only the representations learned with W2V.

2.3.2 Command goal classification. The second pretext task consists in classifying tokens belonging to the *commands* category according to their *goals*. For instance, both *wget* and *curl* primary goal is to download content, while *cd* and *rm* allow file system manipulations.

Since determining command goals is subjective, depending strongly on the application context, we use two different sources to define the labels for the pretext tasks. In the first setup (which we call *general*), we rely on an introductory bash tutorial [1] to define intuitive goals for the commands. The tutorial identifies 9 classes. We obtain 100 labeled commands, ignoring those for which we have no labels.

As a second set of labels (which we call *security*), we rely on the categories presented in [9] to label commands found in SSH honeypot logs. Authors use the Mitre taxonomy [4] to group commands according to the possible reasons attackers would employ them. We have 8 classes, including *Privilege escalation*, *User password manipulation*, and *Reconnaissance*, etc. As before, commands for which we have no class are ignored, leading to a dataset composed by ∼ 60 labeled commands.

2.4 ML models for pretext tasks

To solve the pretext tasks we rely on different ML algorithms: a simple 3 hidden layer feed-forward Neural Network (NN), a K-Nearest Neighbour (KNN) and a Random Forest (RF) classifier. We use models of different types and complexity to observe if the obtained embeddings are consistently informative for all of them, and obtain thus stronger evidences when selecting the best representation.

Given thus a representation obtained with a W2V configuration, we fit a classifier to solve the pretext task. Following good practices in ML, we split the dataset into 70% for training, 20% for testing and 10% for validation, repeating 5 times each experiment and performing each time a grid search with the scope of optimizing the main parameters of each classifier.

To assess whether representations bring any discriminative power, we also include baseline models, consisting of three naive classifiers: (i) a random classifier that uniformly samples a class, (ii) a classifier that always returns the most popular class, and (iii) a classifier that selects a class randomly based on a priori probabilities of the labels.

3 PERFORMANCE ON PRETEXT TASKS

3.1 Token type

3.1.1 CBOW vs Skip-Gram. We first test a limited set of parameter configurations for each model in a preliminary benchmark to choose between Skip-Gram and CBOW W2V models (see the middle columns of Table 1).

Table 2: Macro F1-scores stats for 18 preliminary settings for the two token-based tasks.

	Token Type		Command goal - General	
W2V Model	**CBOW**	**Skip-Gram**	**CBOW**	**Skip-Gram**
min	0.390	**0.771**	0.175	**0.246**
max	0.579	**0.779**	0.257	**0.357**
mean	0.514	**0.775**	0.203	**0.303**
std	0.107	**0.004**	0.022	**0.025**

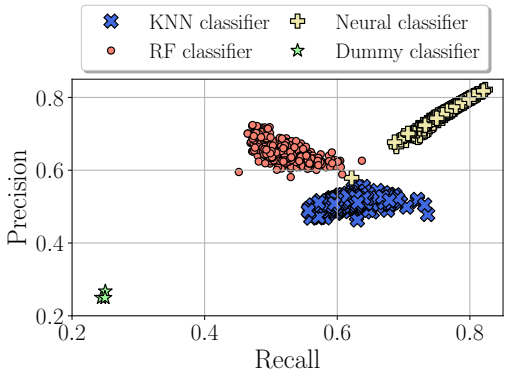

Figure 1: Token type task – macro average precision and recall of different classifiers and the baselines.

Table 2 show the results when using the NN classifier. It reports the minimum, maximum, average and standard deviation of the macro-average F1-score obtained in the experiments. The higher are the results, the better it is.

Focusing on the *token type* column, the representations obtained with the Skip-Gram model outperform those of the CBOW model. The mean macro F1-Score grows from 0.51 to 0.77, and the overall best configuration ("max" line) leads to much higher macro F1-Score when using the Skip-Gram model. These results are consistent also for the other classification models. We thus pick the Skip-Gram model as the best option, and deepen the analysis with additional benchmarks varying other W2V parameters.

3.1.2 Performance across ML models. Figure 1 shows a scatter plot of the macro average precision and recall across classes. Different colors highlight results for the NN, KNN, RF, and three "dummy" baseline classifiers. Each point in the figure is equivalent to a different W2V configuration, with 343 representations tested for each classification alternative. The classifiers face strong class unbalance in this task, since our dataset contains around 3 k commands, 462 flags and 150 k parameters. Whenever possible, we use class weighting when training the models.

First, comparing results of the ML algorithms against the baselines, we conclude that the embeddings do provide informative features for this task. The neural classifier has better average precision and recall than the other algorithms. More interesting, notice how the performance of the same ML algorithm changes significantly as we vary the parameters of W2V. In other words, the representation quality substantially impacts performance, thus supporting our call for good pretext tasks for the choice of the best representation.

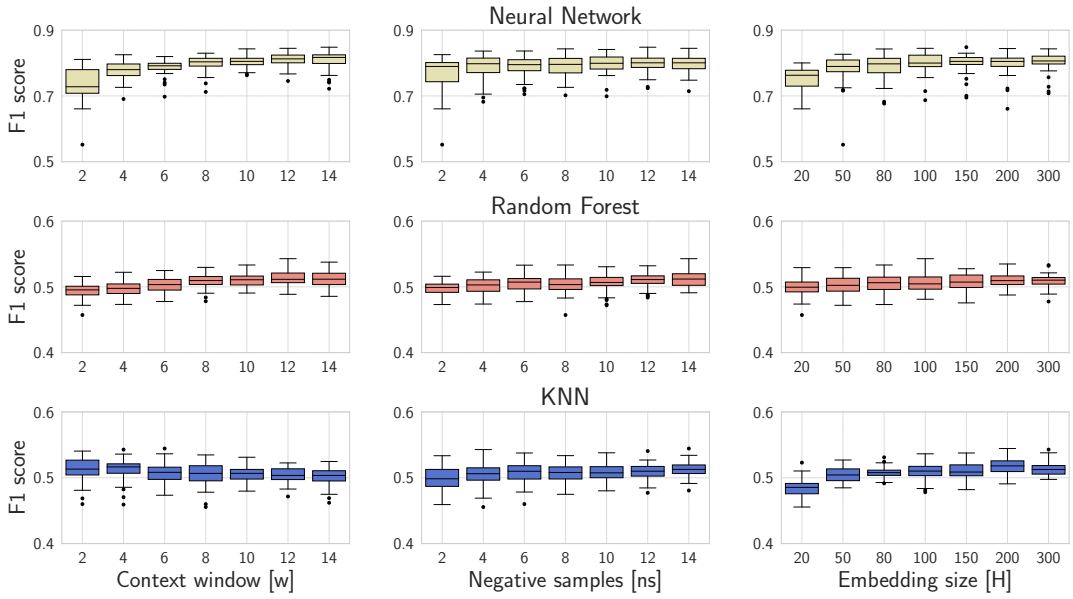

Figure 2: Token type task – impact of W2V parameters on the performance of the ML classifiers.

3.1.3 How to pick the best representation? In order to pick the best representation, we must consider both the fact that a number of parameter configurations are available, as well as the possibility that different classification models may prefer different representations.

To better appreciate the impact of each W2V parameter, we breakdown the performance when varying on parameter at a time. In Figure 2 we plot the F1-score distributions obtained while keeping one W2V parameter fixed and varying the others. Each box in the figure therefore summarizes multiple experiments, in which NN, KNN and RF classifiers are trained with different representations. We focus on context window w, number of negative samples ns, and embedding size H (cf. Table 1, rightmost column).

Focus first on the NN classifier (top plots). It achieves the highest scores when W2V is trained with large *context window*, *negative sample* and *embedding size*. Particularly, the best combination appears with context window $\sim$ 12, negative samples $\sim$ 10 and embedding size $\sim$ 150. The RF and KNN broadly concur with those choices, highlighting the importance of considering multiple ML algorithms in the pretext task to gather evidences for selecting a representation. The only exception refers to the KNN choice regarding the context-window. There, the F1-scores are very similar, suggesting that for the KNN, the context window parameter is not vital for the overall behavior.

3.2 Command goal

Our second pretext task is classifying the command goal. We perform experiments following the same methodology adopted in the previous pretext task. In this case, however, we repeat all experiments twice, using the *general* and *security* class labels introduced in Section 2.3.2. We have $\sim$ 100 and $\sim$ 60 labeled commands for the *general* and for the *security* case, respectively. This limited number of labelled samples discouraged us from training a NN for the task.

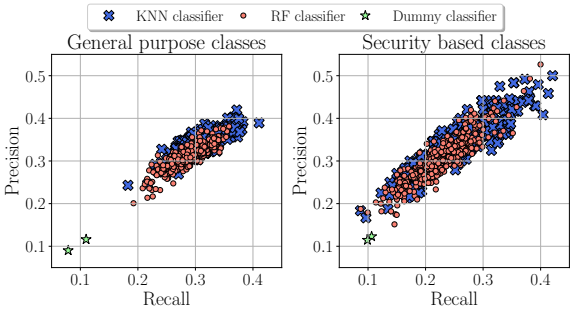

Figure 3: Command goal task – Precision vs Recall for different classifiers and the baselines.

We thus resort only to KNN and RF.

First, check in Table 2 the columns comparing CBOW and Skip-Gram for this pretext task (*general* labels) obtained with the RF classifier. Results still confirm the better quality of representations obtained with Skip-Gram than CBOW, with F1-score of 0.303 against 0.203. As expected, notice how the F1-scores of both versions are much lower with respect to the previous case. While the previous task was based on 3 well-defined labels easily obtained with the grammar parser, the commands goal classifier relies on 8-9 labels that are subjective, thus resulting in a much harder task.

Figure 3 expands the analysis testing more W2V parameters with the Skip-Gram model. Each point is a different embedding. The results confirm that this task is more complex than the previous one – i.e., the performance is generally worse than in Sec 3.1. Yet, we see that classifiers successfully learn how to classify commands, showing results far above the baseline classifiers. Furthermore, it is again evident that different representations produce different

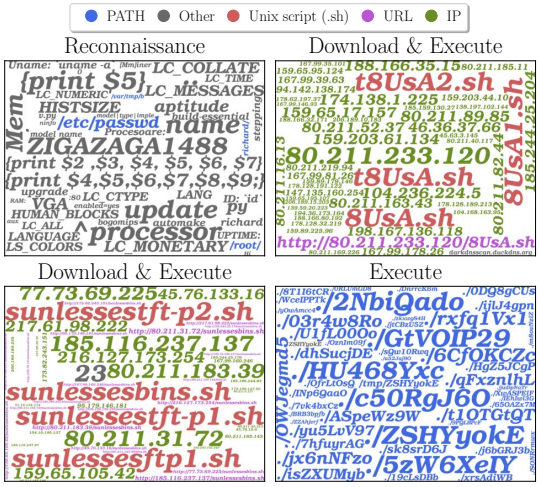

Figure 4: Examples of parameters in the same cluster.

performance, hence underlying the importance of the validation via pretext tasks.

For the sake of brevity, we do not report all results on varying the W2V parameters. However, consistently with the previous case, we obtain the best performance with large *context window*, while we notice small variations when changing *negative sample* and *embedding size*. Thus, the best choices fall in similar ranges also for this pretext task, showing a consistent choice. We hence proceed picking the best parameter previously identified.

4 DOWNSTREAM TASK

4.1 Goals and clustering methodology

Once the learned representations have been successful on the pretext tasks, we can select the best embedding and proceed to solve more challenging downstream tasks. In this section we show a case study where we cluster parameters in bash scripts using their best W2V embeddings as features. One possible application of such clustering is threat analysis. Our assumption is that parameters in similar scripts could be mapped to neighboring regions in the embedding space, and clustering could help to uncover related threats.

We use agglomerative clustering on the parameters embeddings. We select the number of clusters by optimizing the average silhouette, ending up with $\sim$ 800 clusters from the original $\sim 160k$ parameters. Most clusters are small, but we also identify very large clusters composed by more than 10k tokens, still with high silhouette score.

4.2 Interpreting the clusters

Manually going through the clusters, we find that many of them curiously present similarities in the way the parameters are spelled. This is illustrated in Figure 4. See the top-right and bottom-left clusters, in which variants of*8UsA* and *sunlesses* strings are noticeable. We identify parameters in these clusters belonging to standard scripts used by attackers that download particular files for later execution, and we therefore name them *Download & Execute*.

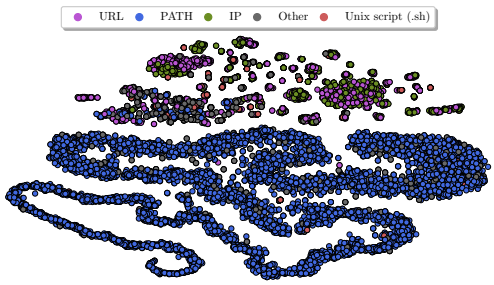

Figure 5: t-SNE plot of the most popular parameter embeddings. Colors mark parameters of the same type.

These scripts' file names, paths and IP addresses are put in the same clusters, even if they belong to different attack sessions and have different spells. Remind that we build our input features by exploiting the co-occurrence patterns of commands, parameters and flags within ssh sessions, i.e., we do not leverage any information about the strings and their semantics. Hence, W2V detects those similarities based only on the parameters' contexts.

Figure 5 extends the analysis by showing a t-SNE plot [25] of the most popular parameters, which we color based on their semantic – e.g., whether they are paths, names of shell scripts, URLs, IP addresses or others. Two clear regions can be identified. On the top of the figure, several points fall in small groups. We confirm that the t-SNE in this region reflects pretty well several small clusters, comprehending *Download & Execute* patterns of Figure 4. In the bottom part of the figure we observe a large number of points, equivalent to paths, shell scripts and others. These points are the same as those in the bottom-right cluster in Figure 4. It contains thousands of random strings, that are file names, which are downloaded using multiple procedures. Yet, as they are used as executable files in later parts of the scripts, the representation has captured their context, mapping all these parameters to a single region of the space.

All in all, results illustrate how our W2V embeddings, whose selection was guided by pretext tasks, carry informative representations that can be exploited in more complex downstream ML tasks.

5 CHALLENGES FOR PRETEXT TASKS

We also illustrate a case of an ultimately poorer pretext task – probably due to limitations of the groundtruth data. Pretext tasks should be indeed avoided whenever they provide little useful information about the representations, as was the case with this task.

5.1 Statement output prediction

This, also domain-specific, pretext task aims at determining whether an attacker's input, which we here name *statement*, produces shell outputs. The pretext task is thus a binary classification problem, with the positive class occurring when either errors or outputs are sent back to attackers.

This problem has the characteristics of a good pretext task: it is a well-defined classification problem, where labels are easy to obtain. To generate the labels, we select a subset of $\sim$ 10 000 inputs observed in our data. We then run them in a sandbox virtual machine (VM).

Each attack session is run starting from a clean VM snapshot, and we register both the *stderr* and *stdout* outputs after each input line. This procedure results in a dataset with 10k samples, 72% of them producing no output.

Notice that the problem is a *statement-based task*, as opposed to the *per-token* tasks discussed in the previous sections. Each statement is a variable-size series of tokens, each represented by its own embedding. We thus need additional steps to handle the complex input features in the pretext classifiers. We follow a basic *sentence embeddings* approach: we encode each statement by weight-averaging all embeddings composing the statement. We use the *tf-idf* [20] as weights to give higher importance to tokens that may be more relevant for the statement.

5.2 Results and discussion

First, naively splitting the statements into training, validation and test sets leads to a suspicious 100% accuracy, regardless of the representation. This happens because many of the statements that produce output are similar sequences of commands, with small variations of (usually random) parameters. For example:

```
echo -e aiwoe9ye9thaemoPh4ei | passwd | bash
sudo echo -e Ahrie2sai2aic0Wohroe | sudo passwd
```

are two statements in our corpus producing similar output. The presence of random parameters results in tokens with high *tf-idf*, which easily uncovers the class label. In other words, the pretext task becomes trivial and/or models are easily biased. The task does not fully exercise the discrimitative power of the representation and, as such, it is a useless pretext task.

We then test more challenging variations in which we exclude all parameters, ending up with ~ 290 statements. Yet, a common pattern emerges: by cancelling the parameters, the sequence representations obtained with weighed *tf-idf* lack discriminative power. The task turns out to be too difficult to be answered using such representations. Not shown for brevity, performance of all ML models results similar to those of the baseline models.

Summarizing, when the pretext task becomes too easy, we cannot compare representations since all of them succeed in the task. Upgrading the difficulty, our representations come short in solving the problem. Finding better tasks for statement-based cases will be our focus in future work.

6 CONCLUSIONS

Self-supervised NLP representation learning techniques can extract rich features from network logs even in the absence of labels. Learned features depend however on the learning technique and its hyperparameters, making it difficult to decide which representations are better suited for downstream ML task, especially when they are unsupervised. We illustrated how carefully designed pretext tasks help objectively choosing the best representation: (i) results show that certain representations consistently give good results across pretext tasks, regardless of the ML model; (ii) our qualitative analysis of clusters obtained using our pipeline confirms the good and useful quality of our selected representations.

REFERENCES

[1] 2022. Bash Command Classification. http://etutorials.org/Linux+systems/how+linux+works/Appendix+A+Command+Classification/.
[2] 2022. bashlex - Python parser for bash. https://github.com/idank/bashlex.
[3] 2022. Honeypot as a Service. https://haas.nic.cz/.
[4] 2022. Mitre Enterprise tactics. https://attack.mitre.org/tactics/enterprise/.
[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.
[6] N. Aussel, Y. Petetin, and S. Chabridon. 2018. Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE Computer Society, Los Alamitos, CA, USA, 237–243. https://doi.org/10.1109/MASCOTS.2018.00031
[7] Pranjal Awasthi, Nishanth Dikkala, and Pritish Kamath. 2022. Do More Negative Samples Necessarily Hurt in Contrastive Learning? *arXiv preprint arXiv:2205.01789* (2022).
[8] Zied Ben Houidi and Dario Rossi. 2022. Neural language models for network configuration: Opportunities and reality check. *Computer Communications* 193 (2022), 118–125. https://doi.org/10.1016/j.comcom.2022.06.035
[9] Matteo Boffa, Giulia Milan, Luca Vassio, Idilio Drago, Marco Mellia, and Zied Ben Houidi. 2022. Towards NLP-based Processing of Honeypot Logs. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 314–321. https://doi.org/10.1109/EuroSPW55150.2022.00038
[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015
[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv:2002.08155* (2020).
[14] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. 2021. DarkVec: automatic analysis of darknet traffic with word embeddings. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 76–89.
[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).
[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).
[18] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. 2016. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249* (2016).
[19] Devi Putri. 2019. Honeypot Cowrie Implementation to Protect SSH Protocol in Ubuntu Server with Visualisation Using Kippo-Graph. *International Journal of Advanced Trends in Computer Science and Engineering* 8 (12 2019), 3200–3207. https://doi.org/10.30534/ijatcse/2019/86862019
[20] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Citeseer, 29–48.
[21] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. 2017. IP2Vec: Learning Similarities Between IP Addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 657–666. https://doi.org/10.1109/ICDMW.2017.93
[22] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Advances in Neural Information Processing Systems* 33 (2020).
[23] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. 2020. Rethinking few-shot image classification: a good embedding is all you need?. In *European Conference on Computer Vision*. Springer, 266–282.
[24] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.
[25] Laurens van der Maaten and Geoffrey Hinton. 2008. Viualizing data using t-SNE. *Journal of Machine Learning Research* 9 (11 2008), 2579–2605.