CS230

Host Similarity on Palo Alto Networks Dataset

Seiji Eicher
Department of Computer Science
Stanford University
keicher@stanford.edu

Abstract

This project set out to learn the a similarity function on JSON host signatures using an autoencoder to learn embeddings, and k-means to perform clustering on the latent space. We saw little success recreating the original host clusters after experimenting with various feature engineering methods and LSTM/Transformer-based variational autoencoders, achieving a max average cluster match probability of .004.

1 Introduction

Identifying malicious hosts is a top priority cybersecurity task, critical to understanding and defending networks against malware. In this paper we investigate methods to exploit structure on the set of host domains in order to create a notion of similarity among the set of hosts that a piece of malware has communicated with. Additionally, we hypothesize that this similarity will be embedded in the services running on a given host. Using these sets of hosts associated with a given piece of malware, we will apply deep neural networks to generate a similarity function in the services space that will allow us to identify and remove clusters of hosts.

The main challenge is encoding the structure of a host services signature as input to the neural network. Due to the inconsistency in fields between the host service signatures, it seems necessary to reduce some fields to an indicator variable. Additionally, more research is needed to determine a way to structure the input so that the fields which are shared among all data points can be processed by the model as natural language. That is to say, a single input to the model would be a vector of strings.

2 Related Work

There have been many applications of deep neural networks to learn a notion of similarity on different spaces, with applications in generating structured drum patterns, molecular similarity, and image generation[1][2][3]. However, relatively little has been published on host similarity work in deep learning; Though analysis of domain names and IP traffic has been done, this is limited in scope to the information provided by a full host signature.[4] On the other hand, sophisticated metrics of similarity among hosts has been completed without the tool of neural networks.[5] Specifically, though this dataset was published in 2018, there are no publicly available papers which explore the proposed problem of host similarity.

3 Dataset

The dataset was obtained from Palo Alto Networks as a part of their paper “Machine Learning in Cyber-Security - Problems, Challenges and Data Sets "[6]. The raw data consist of 136,430 domain connections, grouped by SHA256 hash of the malware that connected to them. Each entry contains:

- The domain with which the malware communicated, the number of different “benign" files communicating with the domain according to Palo Alto Networks threat intelligence
- The number of different malware files communicating with the domain
- The number of different grayware files communicating with the domain (files whose classification is not clear cut/subjective)
- DNS request type
- Response to the DNS request

Additionally, for each of the 19,776 domains that were included in the list of domain connections, there is a “host services signature": the list of services opened on the host and the profile of each service. Our learned clusters of hosts will attempt to match the sets of hosts grouped by malware in the domain connections list. The host services signature is a JSON object with 20 top level keys (and nested objects within), some which are not present on all signatures.

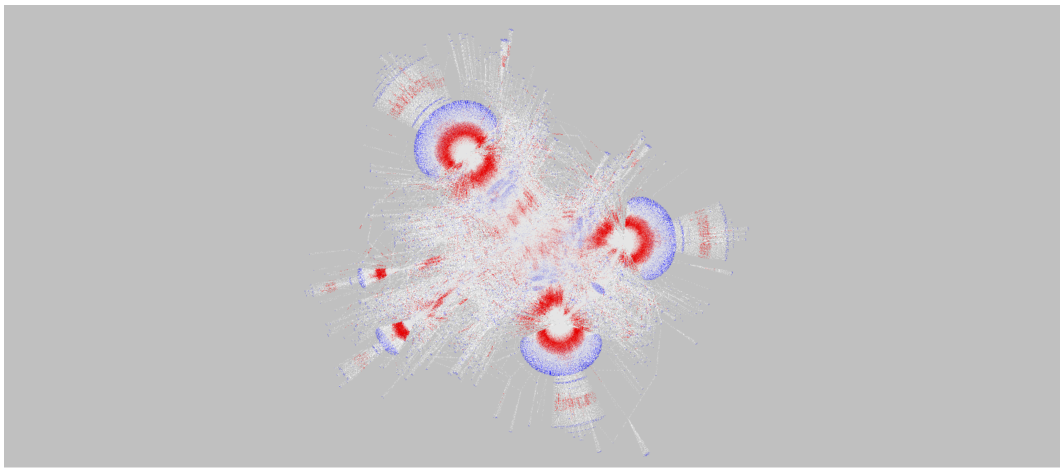

Figure 1: Visualizing the connections between malware and hosts in the dataset using Graphia.[7]

3.1 Clustering

Figure 1 shows malware are associated with hosts as a network. Note that there are three major clusters which correspond to malware interacting with three IP addresses associated with `time.windows.com`, each accounting for roughly 20,000 connections. Otherwise, this graph shows the complexity of the interactions within the dataset; Many malware interact with the same hosts, which can obfuscate hosts that strongly identify a certain malware.

We generated clusters by iterating through the list of domain connections, creating a running list of hosts which had corresponded with each malware. This step associated 33,298 unique malware with a total of 14442 unique signatures, which shows that there was missing data in the list of domain connections and that many malware interact with common hosts. Figure 8 shows a histogram of the cluster sizes, showing that the vast majority of malware connect to about 4 hosts.

4 Methods

4.1 Feature Engineering

For baseline input representation of the JSON-encoded host signature, we depth-first flatten the JSON dictionary into a list of strings, with keys followed by values. Then, we create a corpus based on the entire dataset and replace each string with its integer id from the corpus. This is then fed into an embedding layer. This approach results in sequences that have lengths around 500-600 tokens, with a maximum around 200, and a corpus of size 140,000, since it encodes every string in every signature.

Next, we limit the number of fields from the host signature to just those that are present in all signatures in the dataset. Additionally, we remove the `'data'` field, since it contributes a lot of variance to the length of the signatures and the content itself would not be relevant to the clustering task. This results in a list of 16 keys. This is our simplest representation, which follows the same regime as above, but with a much smaller feature set. The lengths of the sequences range from 52-58, with a corpus length of 49081.

Finally, we experiment with a combination of the previous common keys approach with a character-level VAE in order to compare key's actual values rather than simply their relative value (whether or not they match). This results in signatures with length distributed as in figure 5.

4.2 Neural Network

In order to approximate a similarity function on the host signature space, we used a variational autoencoder (VAE) to send the variable-length strings of tokens to a fixed-size latent space. Within the variational autoencoder framework, we experimented with both LSTMs and basic Transformer blocks as encoder/decoder pairs.

The loss function used in the VAE for both architectures was (1), where $\mathcal{L}$ denotes the reconstruction loss and KL is the KL divergence, a measure of difference between two probability distributions, summed over each dimension j of the embedding space.

$$\mathcal{L}(x, \hat{x}) + \sum_j \mathrm{KL}\left(q_j(z|x)||p(z)\right) \tag{1}$$

The first architecture used bidirectional Long Short-Term Memory networks for encoding and decoding within the VAE [8]. This model had a variable number of parameters due to the embedding layer, which generated embeddings from the 0-padded tokenized sequences depending on the size of the corpus. All LSTM-based models used a 64-dimension embedding space, sampled from length 128 LSTM sequence outputs.

We also experimented with using a transformer block with the common keys, character value encoding input representation. This model generated length 256 input embeddings for each of the tokens in the input string according to token and position, which fed into a transformer block with output length 256. Next, global average pooling was applied to generate a sequence of length 256 from which 128 points were sampled for the VAE step.

All models were built using Tensorflow.Keras and trained to convergence using the Adam optimizer, with a validation split of .1 and batch size of 64.[9][10] The training sequences were padded to the maximum sequence length, then shuffled for a train/test split by `scikit-learn`'s `train_test_split()` and `pad_sequences()`.[11].

4.3 Evaluation

In total, 14442 unique signatures were found in the domain links database. The autoencoder was trained with a 90/10 train/test split, where per-epoch validation was done on a random .1 proportion of the training data. The test set was thus only predicted on for the purpose of evaluating clusters.

The set of domain connections provides ground truth similarity clusters against which to compare our clustering results. The premise is that hosts that communicate with the same malware should have similar host signatures, we should see high probability that that the dev set hosts' cluster placements will be similar to their ground truth clusters. Therefore, our evaluation metric is the average probability that we will see a related host in the cluster we generate. That is, given that a host x was placed in a certain cluster, we are interested in proportion of the cluster's hosts that communicated with a malware that communicated with x.

To begin, we run the k-means algorithm on the predicted encodings of the training set, setting k to the number of clusters in the original dataset. To evaluate the performance of the similarity function, we predict the clusters of hosts in the dev set and compute the previously mentioned metric for each sample in the dev set, returning the average of these results.

5 Results

Model	Avg. Cluster Match Prob.	Enc. Params	Dec. Params	Sec./Epoch
All Keys	0.001 29	6 949 824	66 177	1600
Common Key Strings	0.000 88	3 223 744	66 177	45
Hybrid w/ Char Encoding	0.001 11	90 048	66 177	356
Transformer	0.004	1 668 608	396 289	93

Figure 2: Caption

5.1 Using all strings

Using all the strings in a host signature as input representation produced an average cluster match probability of .00129, which is likely due to random chance of a host signature falling into the same cluster as another. Figure 3 shows the latent space of the “all strings" method, which shows seemingly random assignment to clusters. Since the corpus was of size 107301, a much larger model would likely have been necessary to encompass this much data, beyond the scope of the allocated resources.

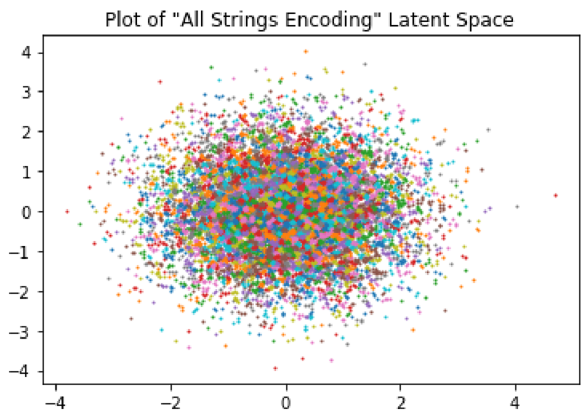

Figure 3: A PCA reduced scatter plot of the latent space, color coded by k-means cluster.

5.2 Common keys only

The common keys with value string input representation achieved the lowest performance out of the models tested, with average cluster match probability .00088. This could be due to the fact that the “all strings" method encodes some information about which fields are present, even though it doesn't know the meaning of their values. In this case, the host signature fields are all the same and the model can only gain a notion of similarity if the values match exactly, likely contributing to its slightly worse performance.

5.3 Common keys with character-level encoding

Using a hybrid of reducing the host signature to common keys and encoding the values by character was the most promising input representation, since it would seem to give the model the most structure to work off of while still allowing comparison between values (for example, comparing similar latitude/longitude pairs), but this method too achieved a limited average cluster match probability at .00111. This may be evidence toward the complexity of the network connections (i.e. the problem itself), or a need for larger models and more compute to absorb the data.

5.4 Transformer-based VAE with Character-level encoding

Though the transformer-based VAE had many more parameters than the hyrid LSTM-based model and represents the state-of-the-art, it did not learn, despite hyperparameter tuning. This may represent a need for an even deeper model, or a bug in the implementation, since this was an unexpected result. The transformer model displayed robust training time despite the higher number of parameters, making it promising for future work.

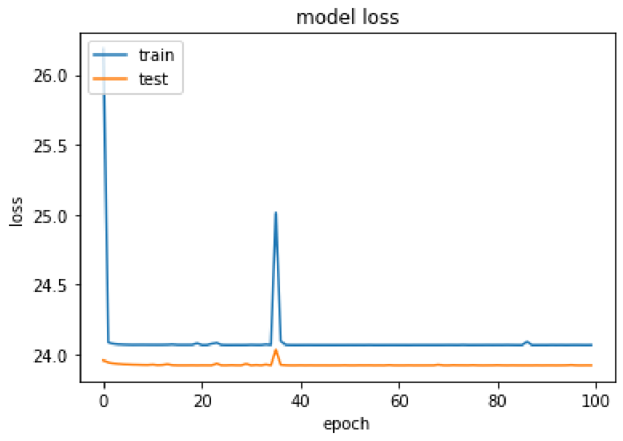

Figure 4: Transformer-based VAE loss plot.

6 Conclusion

Limited by time and compute resources, a larger VAE model might be promising. Additionally, further pruning of hosts that see a large multiplicity of connections, such as `time.windows.com` could remove many connections that do not contribute information to the clustering task.

Another direction that this task could move forward is pivoting to training a discriminator model to distinguish between hosts, rather than using the autoencoder to organize the latent space. This would allow incorporating the domain connections information into the training process, e.g. using triplet loss to decide if two hosts were in the same cluster, could incorporate further information into the model. Furthermore, this might allow further prediction, on the host or cluster level, of a cluster as benign, malware, or grayware using the statistics from the domain links file.

References

[1] Yihua Chen, Eric K. Garcia, Maya R. Gupta, Ali Rahimi, and Luca Cazzanti. Similarity-based Classification: Concepts and Algorithms. *Journal of Machine Learning Research*, 10(27):747–776, 2009.

[2] Alexey Dosovitskiy and Thomas Brox. Generating Images with Perceptual Similarity Metrics based on Deep Networks. *arXiv:1602.02644 [cs]*, February 2016. arXiv: 1602.02644.

[3] I-Chieh Wei, Chih-Wei Wu, and Li Su. GENERATING STRUCTURED DRUM PATTERN USING VARIATIONAL AUTOENCODER AND SELF-SIMILARITY MATRIX. page 8, 2019.

[4] Tao-Wei Chiou, Shi-Chun Tsai, and Yi-Bing Lin. Network security management with traffic pattern clustering. *Soft Computing*, 18(9):1757–1770, September 2014.

[5] Scott Coull, Fabian Monrose, and Michael Bailey. *On Measuring the Similarity of Network Hosts: Pitfalls, New Metrics, and Empirical Analyses*. January 2011.

[6] Idan Amit, John Matherly, William Hewlett, Zhi Xu, Yinnon Meshi, and Yigal Weinberger. Machine learning in cyber-security - problems, challenges and data sets. 2018.

[7] Tom C. Freeman, Sebastian Horsewell, Anirudh Patir, Josh Harling-Lee, Tim Regan, Barbara B. Shih, James Prendergast, David A. Hume, and Tim Angus. Graphia: A platform for the graph-based visualisation and analysis of complex data. *bioRxiv*, 2020.

[8] Jason Brownlee. Encoder-decoder recurrent neural network models for neural machine translation, Aug 2019.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[10] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

7 Appendix

7.1 Data plots

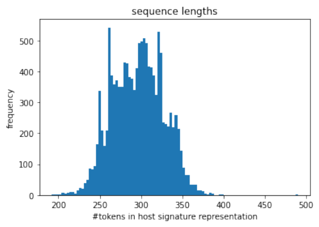

Figure 5: Distribution of lengths for common keys with character-level encoding.

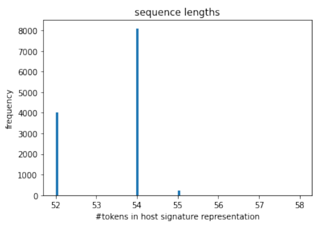

Figure 6: Distribution of lengths for string-based common key encoding.

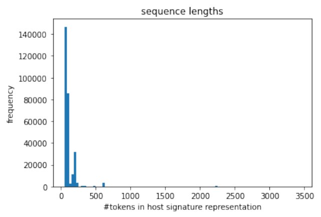

Figure 7: Distribution of lengths for all strings encoding.

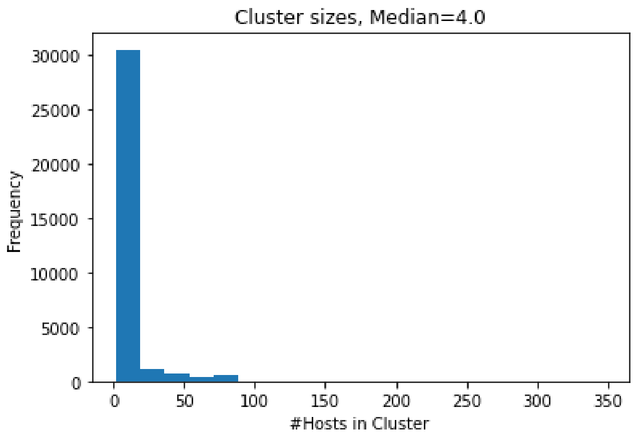

Figure 8: A histogram of the number of hosts associated with a piece of malware.