A Benchmark of Unsupervised Off-The-Shelf Anomaly Detection Methods on ADFA-LD

Julia Rolland
Université de Technologie de Belfort-Montbéliard
Belfort, France
julia.rolland@utbm.fr

Quentin Fournier
Polytechnique Montréal
Quebec H3T 1J4
quentin.fournier@polymtl.ca

Daniel Aloise
Polytechnique Montréal
Quebec H3T 1J4
daniel.aloise@polymtl.ca

Abstract

Programs communicate with the kernel of an operating system with low-level events called system calls. Consequently, system calls encapsulate the system behaviour and are a great source of data to detect anomalies. However, the manual evaluation of system call sequences is highly inefficient as the data generated is too large. Instead, machine learning techniques allow processing those sequences automatically. Although the manual inspection may reveal unknown irregularities that machines cannot extract themselves, the human labour cost is prohibitive in practice. This work explores unsupervised approaches from natural language processing to learn fixed-size representations of the sequences, namely bag-of-word, TF-IDF, LDA, and Skipgram. They are detecting outliers with unsupervised off-the-shelf methods, namely cosine similarity, k-NN, isolation forests, and one-class SVM. Experiments were conducted on the recent ADFA-LD dataset collected on a modern operating system (Linux 11.04).

I. Introduction

Tracing is a technique that utilizes logs to record low-level information about a program's execution. Tracing produces a large quantity of data called traces, corresponding to a sequence of low-level events. Most notably, traces are used for debugging and diagnosing problems. This paper considers traces of kernel events called system calls or syscall. System calls correspond to requests from programs to the operating system's kernel on which they are executed. In other words, system calls create, execute and ensure communication between the operating system and processes. There are five categories of system calls: process control, file management, device management, information maintenance and communication. System call traces are effective for anomaly detection as they capture most of the system behaviours. This paper aims to compare off-the-shelf unsupervised outlier detection techniques to detect anomalies. However, those techniques typically require fixed-size input instead of variable-size sequence.

An important property of system call sequences is their semantic and syntax, similarly to natural languages. For instance, a file must be opened before being closed, hence the system call `close` cannot be called before the system call `open` for a given file, and the sequence "`open`, `write`, `close`" has a clear meaning. Consequently, methods developed for natural language should also apply to system call sequences.

Natural Language Processing (NLP) is a discipline that focuses on the understanding, manipulation and generation of natural language by machines. That means that machines can encapsulate the semantic and syntax of human language to reproduce or classify it. There are two essential aspects of NLP. First, the "linguistic" part, which consists of preprocessing and transforming the input information into an exploitable dataset. Second, the "machine learning" or "Data Science" part, which focuses on the application of Machine Learning or Deep Learning models to this dataset. The goal of NLP is to model language with the same level as humans (not necessarily in the same way, but with the same quality, fine-grained nuances).

The proposed methodology to detect anomalous traces is to learn fixed-size representations of system calls sequences (§II-A) and apply off-the-shelf outlier detection methods on those representations (§II-B). The methodology is applied to the ADFA-LD dataset, which is recent, labelled, and popular (§III), and the results are discussed providing insight on each approach (§IV).

II. Methodology

Supervised learning algorithms can solve the anomaly detection problem if we have information on anomalous behaviour beforehand. Known anomalies are typically addressed and fixed immediately. However, in our case, we want to detect new anomalies without waiting for some feedback. That is why we use unsupervised algorithms to detect outliers within our dataset.

A. Representation

Our dataset comprises variable-size sequences of integers corresponding to system call names. We use off-the-self outlier detection methods that require fixed-size input. Therefore, we apply machine learning and deep learning function from natural language processing to learn fixed-size representations of the sequences.

1) Bag of Word: A Bag-of-Word model, also called BoW, is a straightforward technique of NLP used for information retrieval. In linguistics, it was first cited in 1954 by Harris [1], and it was applied later in computer vision to sort images by processing them as words. Technically, it calculates the number of occurrences of each word in a document. In our case, it measures the count of each system call in the sequence. To do that, it builds a vocabulary of all the system calls passed through the algorithm during the training phase. Each variable size-sequence is represented as a vector of the size of the vocabulary. The drawback of this representation method is that it dismisses many factors such as order or structure. We could use a bigram or trigram instead of a one-gram BoW to learn more about the arrangement to remedy that problem partially.

2) TF-IDF: The Term Frequency Inverse Document Frequency, noted $\mathrm{tf\text{-}idf}$, is a method to determine the importance of a word throughout a set of documents. To give the intuition behind $\mathrm{tf\text{-}idf}$, let us consider a biology book, which is a collection of paragraphs. The word *the* appears in virtually all paragraphs. Therefore *the* does not describe the paragraph, and its influence should be reduced. Instead, the word *tardigrade* appears in fewer paragraphs. Therefore, it is useful to describe the paragraph, and its influence should be increased. The Term Frequency Inverse Document Frequency defines the power of a term by considering its influence in a single document and scaling it by its weight across all documents. The $\mathrm{tf\text{-}idf}$ is the product of the *term frequency* and the *inverse document frequency*. There are various mathematical ways to compute these two values.

The *term frequency*, noted $\mathrm{tf}(t, d)$, is the frequency of the system call t obtained by:

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

where $f_{t,d}$ is the number of times that system call t occurs in sequences d. The *inverse document frequency*, noted $\mathrm{idf}(t, D)$, measures the weight of the system call within the sequence and is computed as follows:

$$\mathrm{idf}(t, D) = log \frac{N}{|\{d \in D : t \in d\}|} \tag{2}$$

with N the number of sequences in the dataset $N = |D|$ and $|\{d \in D : t \in d\}|$ the number of sequences where the system call t appears. Since the ratio of idf is always greater or equal to 1, we use the log function to delimit it: idf is then greater or equal to 0. Finally, $\mathrm{tf\text{-}idf}$ is then determined by:

$$\mathrm{tf\text{-}idf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D) \tag{3}$$

The score value of $\mathrm{tf\text{-}idf}$ is always between 0 and 1 due to idf. The closer it gets to 1, the rarer the term is in all the collection of documents. This method helps to adjust the importance of words based on their rarity.

3) LDA: The Latent Dirichlet Allocation, also called LDA, is a representation employed to find the most frequent word in a corpus. It was first applied in the context of population genetics but was later utilised in machine learning by Blei et al. [2] in 2003. It is based on the principle that documents with the same topics employ identical or similar words. It can then be used to represent a document by its subjects. The principle is to go through each document and randomly assign each word in the document to one of k topics (k is chosen beforehand). For each word of each document, the methods computes two probabilities. The first one corresponds to the share of words in a document d that are appointed to the topic t: $p(t|d)$. The second is the proportion of assignments to topic t over all documents that come from this word w: $p(w|t)$. Then, the probability of the word w belonging to topic t is:

$$p(w \in t) = p(t|d) * p(w|t) \tag{4}$$

where w is the word, d is the document and t is the topic. Then depending on these probabilities, we take the best ones to find the topic of documents.

4) Skipgram: One of the simplest ways to represent words is called one-hot encoding vectors. Given a vocabulary of n words, the one-hot encoding of the i-th word is a vector of dimension n comprising 0 except for the i-th position that takes the value 1. However, the one-hot encoding approach has shortcomings: (1) if the vocabulary size is big, algorithms have to deal with large vectors, and (2) they fail to capture the semantic and similarity of words as all one-hot encoding vectors are perpendicular. Instead, we use another technique called embedding. Word embedding is the method that consists of representing a word in a more compact vector that encapsulates the semantic. Converting the word to vectors of numbers makes it easier to compare the similarity in other words.

Our fourth representation is a neural network called word2vec that is widely popular thanks to the quality and the compactness of learned embeddings. This model can be implemented either using Continuous Bag-of-Word (CBoW) or the Skipgram algorithm proposed in 2013 by Mikolov et al. [3]. The two algorithms are very similar: CBoW learns the embedding of predicting the words (output) knowing their context (input), while Skipgram learns the embedding of predicting the context (output) of each word (input). Both methods use a defined window of neighbouring words (see Figure 1). For this project, word2vec is implemented with the Skipgram algorithm.

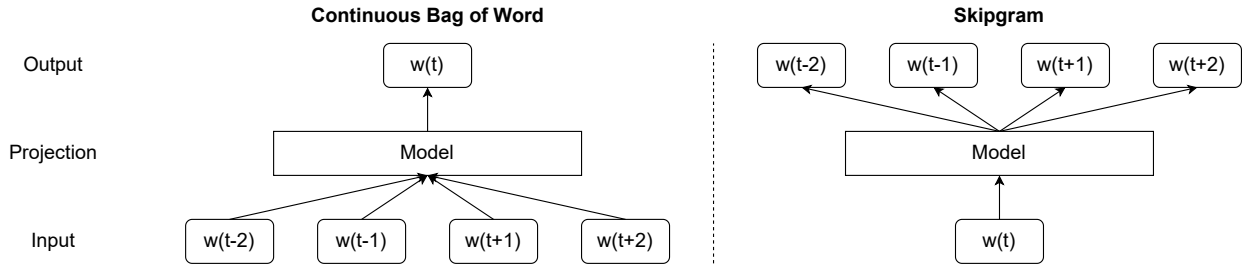


Fig. 1. (Left) Continuous Bag-of-Word model. (Right) Skipgram model.

Word2vec is composed, like other neural networks, of an input layer, a hidden layer and an output layer as we can see in 1. In the input layer, we pass the target syscall and in the output layer, the algorithm gives the context words of the targeted syscall. The hidden layer is more complex. It is where all the functions of the model are applied to the input to obtain the output. First, we convert the syscall to a one-hot encoding. Then, we multiply it by two dense matrices that denote word embeddings for the target *W1* and the context words *W2*. They are compounds of real numbers and of size $W_1 \in \mathbb{R}^{V \times E}$ and $W_2 \in \mathbb{R}^{E \times V}$ where *V* is the size of vocabulary and *E* the dimension of embeddings such as $E \ll V$. This step is shown by 5.

$$\underbrace{\begin{bmatrix}0\\0\\\vdots\\0\\1\\0\\0\end{bmatrix}}_{\text{Input}\in\mathbb{1}^V} \cdot \underbrace{\begin{bmatrix}0.361 & 0.642 & -0.919 & -0.747 & -0.252\\0.357 & 0.219 & 0.808 & -0.972 & 0.184\\2.324 & -0.863 & -0.708 & 1.911 & -0.601\\\vdots & \vdots & \vdots & \vdots & \vdots\\0.263 & 0.620 & 0.507 & 0.944 & -0.967\\0.504 & -0.325 & 1.131 & 1.762 & -0.410\\0.156 & 0.274 & 0.487 & 0.199 & 0.884\end{bmatrix}}_{W_1\in\mathbb{R}^{V\times E}} \cdot \underbrace{\begin{bmatrix}1.494 & -0.150 & 1.623 & \dots & 0.066 & -0.781 & 0.416\\-0.852 & 0.454 & -0.696 & \dots & -0.250 & 0.275 & 1.248\\-0.150 & 1.137 & -0.123 & \dots & 0.188 & -2.096 & -0.893\\-0.143 & 0.353 & -0.231 & \dots & -0.237 & 0.625 & -0.695\\-0.161 & -1.194 & 1.172 & \dots & -0.973 & 0.459 & 0.510\end{bmatrix}}_{W_2\in\mathbb{R}^{E\times V}} \tag{5}$$

Next, the prediction vector y_pred is obtained by passing the output of the equation 5 through a Softmax function, which transforms the vector of *K* real values into a vector of *K* probabilities. It is typically the last activation function of neural networks for multi-class classification. It was first cited in machine learning by Bridle [4] in 1990, but it is an older concept. The Softmax equation is:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=0}^{K} e^{x_j}} \tag{6}$$

where x is the vector obtained after multiplying the input vector by the two embeddings matrices, x_i are all the values of the input vector (i ranges between 1 and K) and K is the size of the vocabulary. The bottom term of the equation is the normalization process that ensures that all the vector sum to 1.

In machine learning, the loss function *l(y, y')* measures how well the model is performing by evaluating the difference between the prediction *y'* and the actual answer *y*. The smaller the loss is, the more effective the model is. We followed the original method and paired the Softmax function with the negative log-likelihood (NLL) given by the following formula:

$$\mathrm{l} = \sum -log(p(X)) \tag{7}$$

where X corresponds to the correction predictions, $p(X)$ is the probability of X in $\mathrm{y\_pred}$.

Two important parameters of neural networks are the epoch and the batch size. The first one corresponds to the number of times the entire training data has been through the algorithm. The second one is the dimension of the data passed through the model. Batches help save memory and updating more often. When they are well configured, they help the model run smoothly and quicker. They also serve as a marker when we execute the model.

In deep learning algorithms, obtaining results often depends on the size of your dataset. Even when the model is straightforward, passing through the algorithm a billion times takes time. That is why, nowadays, we often see optimizers. They are used to minimize the loss function of the model. To do that, we use calculus concepts such as derivatives of function: they can help minimize a function $f(x)$ by small moves x in the opposite sign of the derivative. This technique is called gradient descent by Cauchy (1847). It is used on all the data. Batch gradient descent computes the gradient on a small subset and stochastic gradient descent computes the gradient of a single element.

In our function, we use the optimizer *Adam*. It is an extension of the stochastic gradient descent introduced in 2015 by Kingma and Ba [5]. Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirements. It calculates individual learning rates for different parameters from estimates of first and second moments of the gradients. Adam does not work like the stochastic gradient descent. It adapts the learning rate for each of the parameters. The name Adam is derived from adaptive moment estimation. Some of the benefits of the optimizer are easy to implement, reduce computation time, takes less memory, fit for problems with large data/parameters, and hyper-parameters are intuitive and do not need much tuning.

Training models in machine learning and deep learning takes time and resources. If the model is not trained enough, the model will not capture the data structure. If the model is trained too much, it will overfit on the training sample and produce poor results on the test set. The model will stop generalizing and start to learn the statistical noise in the training dataset. It will make the model less useful in predicting new unseen data. We have to train the model long enough to learn the mapping from inputs to outputs. We tried different approaches (number of epochs, size of batches) to solve this problem. However, these were unique solutions that worked only in our case: it was inefficient. Sometimes, when we train a model, we realize that the values obtained on the validation set do not improve and start to regress at a point in the training. So to optimize our training time, we implemented Early Stopping. It is a regularisation process applied in deep neural networks. When updates of parameters no longer improve on a validation set, it stops the training. It is based on the value of the average loss. When it stops getting smaller, we stop the training.

B. Outlier detection

To detect the outliers within our dataset, we have implemented five different methods. They all follow the same template to make it easier to switch between the functions. We train them on the training set comprising only normal sequences. Once the model is trained, we perform predictions to obtain an outlier score for every sequence. In order to determine if the predicted system call is an anomaly or not, we use a threshold. It is a comparison parameter: if the outlier score is lower than it, then it is considered as normal but if the outlier score is higher, then it is considered as an outlier. We try many different combinations of parameters: threshold and parameters such as the number of neighbours. And then, we keep the set that gives the best results on the validation set.

1) Cosine Similarity: We decided to implement cosine similarity as our first detection method as it is simple and efficient. The geometry-based method assesses the parallelism between two vectors by measuring the cosine of their angle thanks to the formula:

$$\cos(x, y) = cos\frac{x \cdot y}{\|x\| * \|y\|} \tag{8}$$

where x and y are vectors of length $\|x\|$ and $\|y\|$, respectively. If two vectors are parallel, then they are alike and their cosine similarity is equal to 1. If the vectors are perpendicular, they are different and their cosine similarity is equal to 0.

We apply the method to the representations of the system call sequences. We define the outlier score to be:

$$score = 1 - \frac{1}{n}\sum_{i}^{n} cos(x, y_i) \tag{9}$$

where $cos(x, y_i)$ is the cosine similarity between x and y.

2) k-Nearest Neighbour: Our second detection method is k-nearest neighbour (k-NN). This technique can be used for both supervised and, like our case, unsupervised machine learning algorithms. The parameter k corresponds to the number of points that need to be considered a neighbour. Our function tests a range of k to find the value that gives the best F1 score. The outlier score of a sample is indicated by the distance to its neighbours. In particular, we considered three functions: *exact*, *mean* and *hmean*:

- exact: the outlier score of any data point is equal to its k-th nearest neighbour distance to the rest of the set. The distance between the point and its neighbour is computed like a 2-norm distance (Euclidean):

$$\mathrm{d} = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{10}$$

  where x_i is the point and y_i its neighbour.
- mean: the outlier score of any data point is equal to its average distance to its k nearest neighbours in the set. The mean is computed like:

$$\mathrm{mean} = \frac{1}{k}\sum_{i=1}^{k} d_i \tag{11}$$

  where k is the number of neighbours and d_i is the distance to its k neighbours.

- hmean: the outlier score of any data point is equal to the harmonic mean of its distances to its k nearest neighbours in the set. Harmonic mean is computed like:

$$\mathrm{hmean} = \frac{k}{\sum_{i=1}^{k} 1/d_i} \tag{12}$$

where k is the number of neighbours and d_i is the distance to its k neighbours.

3) DBSCAN: Density-Based Spatial Clustering of Application with Noise (DBSCAN) is a popular algorithm to perform clustering introduced by Ester et al. [6] in 1996. It has two sensitive parameters: ϵ the minimum distance for two points to be considered neighbours and m the minimum number of samples to create a cluster. Our function tests a range of values for both and finds which combination gives the best F1 score. To find a cluster, the model starts with a point and fetches all points close in space with respect to ϵ and m. It finds the neighbours with ϵ and create a cluster only if there are at least m neighbours. Otherwise, the point is considered as noise. The particularity of DBSCAN is that it does not train and then effectuates the predictions. So we submitted train and valid collections to tune the hyperparameters. Then we provided train and test collections for evaluation. DBSCAN is our only method that does not give a score. Depending on their localization in the space, the model labels the system call sequences. The label corresponds either to the clusters' number or is considered as noise and takes the value -1. We consider all the noise syscalls as outliers.

4) Isolation Forest: Isolation Forest is a tree-based machine learning algorithm. It works differently from the other unsupervised method. Indeed, it isolates the outliers directly from the dataset. Depending on the threshold, it divides our dataset into two recursively. In the end, it filters the data which took fewer steps than others to be isolated. Since we work with a tree, the number of splitting needed to set apart a point is equivalent to the path length from the root to the point's node. Random partitioning is known to produce a shorter path for an anomaly. The score corresponds to the average of splitting for all the random trees in the forest. Like the cosine similarity, the score is given between 0 and 1. We have to reverse the average path length to obtain an outlier score.

5) One Class SVM: One Class Support Vector Model (SVM) is our last detection outlier method. As the four others, it is an unsupervised machine learning algorithm. It provides significant accuracy with less computation power. It recognizes patterns within a dataset. It is used when the dataset comprises mostly normal data and only a few anomalies. It separates the data into two with a hyperplane. Then it categories the system call sequences. If they are normal, then they are on one side. Else, they are put on the other. It makes predictions by assigning a side for each syscall. The model is trained only on normal data. The score needs to be reversed like the other method.

III. ADFA-LD Dataset

Machine learning algorithms typically require large datasets. Therefore, several methods have been considered to generate large datasets for anomaly detection in computer systems. One approach is to use log files. However, this technique proved ineffective because of three major weaknesses: (1) log files portray interpreted data since they are generated while monitoring system activities, (2) log files do not represent the data according to their importance since crucial data is often flooded by unimportant information, and (3) managing and creating log files is done ad-hoc. These difficulties have been addressed by Creech and Hu [7], Bacon [8] and Wang and Hu [9]. Accordingly, an alternative based on system call traces was introduced, which allowed generating a large volume of data. Plenty of tracing datasets was created based on different operating systems, many of which have already been experimented on. Therefore, we might think we have a choice as to which data collection we will use for our research. However, this is not the case. Indeed, not all datasets are publicly available and labelled – labelling traces requires tremendous manual labour. This is why datasets like UNM and KDD98 are still widely used in scientific research: KDD98 remains a benchmark in this field because it can be manipulated immediately. However, they are handled with caution and scepticism due to their poor representation of modern operating systems.

The collection of data we will use during this internship is called ADFA Linux dataset (ADFA-LD). The dataset was created as a part of research on HIDS (Host-Based Intrusion Detection System) by Creech and Hu [7] from the Australian Defense Academy. ADFA is better than UNM and KDD98 as it was developed on a recent operating system (Ubuntu Linux version 11.04) to resist contemporary hacking methods. Furthermore, the collection of data is publicly available and labelled.

ADFA-LD comprises 833 normal traces for training, 4,372 normal traces for evaluating and 6 different attack sets (see Table I). Supported attack types are web-based exploitation, simulated social engineering, poisoned executables, distanced triggered vulnerabilities, remote password brute force raids and system manipulation.

Although this dataset is relatively recent, there are still some drawbacks. First, it is not big enough for large models. In particular, NLP models usually require massive datasets. For instance, GPT-3 [10] use more than 500 GB of processed text, that is, hundreds of billions of tokens. Second, there is no argument or timestamp: ADFA-LD only consists of sequences of integers corresponding to system call names.

TABLE I
TRACES DISTRIBUTION IN ADFA-LD DATASET

Data Type		Traces
Normal Data	Training Data	833
	Validation Data	4372
Attack Data	Adduser	91
	Hydra FTP	162
	Hydra SSH	176
	Java Meterpreter	124
	Meterpreter	75
	Web Shell	118

IV. PRESENTATION OF THE RESULTS

A. Results

For each representation of our dataset, we tested them with our different models and collected the results of the different metrics: accuracy, precision, recall, F1 score, and area under the ROC curve (AUROC).

The area under the Receiver Operating Characteristic (AUROC) is a performance metric used to evaluate classification models. AUROC tells whether a model is able to correctly rank data. The worst AUROC is *0.5*: it corresponds to a random model. The best AUROC is *1.0*: it corresponds to a perfect classifier. When AUROC gets greater than 0.7, it starts to be a good performance. The AUROC is calculated as the area under the ROC curve. A ROC curve displays the trade-off between true positive rate (TPR) and false positive rate (FPR) across different thresholds. These parameters can be obtained thanks to the decision matrix. In table IV-A, we detailed all the AUROC obtained in this internship. As DBSCAN does not provide a score, we cannot calculate its AUROC.

TABLE II
AUROC ON THE TEST SET OF EACH COMBINATION OF REPRESENTATIONS AND OUTLIER DETECTION METHODS. BOLD RESULTS DENOTE THE BEST REPRESENTATION SCORE AND UNDERLINED RESULTS DENOTE THE BEST OUTLIER DETECTION METHOD.

AUROC	Size	Cos_Sim	kNN - Exact	kNN - Mean	kNN - H mean	DBSCAN	Isolation Forest	OneClass SVM
Bow	341	72.86%	64.48%	64.48%	64.48%	X	48.36%	56.39%
TF-IDF	341	**74.62%**	76.30%	76.30%	76.25%	X	51.06%	**75.77%**
LDA	5	64.39%	59.47%	59.48%	58.95%	X	57.30%	49.49%
	10	73.17%	71.77%	71.77%	70.39%	X	72.07%	71.46%
	20	71.24%	75.45%	75.45%	75.45%	X	**74.89%**	69.01%
Skipgram Sum	5	65.95%	61.68%	61.68%	61.68%	X	52.25%	50.62%
	10	72.28%	60.08%	60.08%	58.21%	X	54.71%	53.86%
	20	72.98%	62.67%	61.23%	60.62%	X	55.38%	56.50%
Skipgram Mean	5	65.95%	**81.19%**	**81.04%**	**80.66%**	X	73.11%	62.67%
	10	72.28%	73.88%	74.39%	73.92%	X	74.60%	67.77%
	20	72.98%	76.95%	76.44%	76.95%	X	73.73%	70.83%

Based on table IV-A, different combinations give good results. However, the combination of representation and outlier detection method that gives the best AUROC is Skipgram Mean 5 and kNN Exact. The best AUROC is 81.19%.

In our case, the most critical parameter is the F1 score. As a matter of fact, the F1 score gives the same weight to precision and recall scores: it is the harmonic mean of the two. So gives information about the predictions' quality of the models. In anomaly detection problems, the most important thing is not to miss an issue. Hence, even if the machine suspects more normal system calls, it does not matter as much as to suspect fewer system call sequences but miss many anomalies.

For each representation (line), the best result is underlined. For each model (column), the best result is in bold. When a result is bold and underlined, it is the best combination of representation and model.

The table IV-A represents the F1 score obtained during the internship. Based on the results. there are fives best combinations.

- Bag of word and Cosine Similarity gives an F1 score of 49.72%.
- Word2vec (Skipgram, Sum, embedding of 5) and Exact kNN gives an F1 score of 56.68%.
- Word2vec (Skipgram, Mean, embedding of 5) and Average kNN gives an F1 score of 56.45%.
- LDA-20 and Isolation Forest give an F1 score of 50.35%.
- TF-IDF and One Class SVM gives an F1 score of 45.37%.

The best F1 score obtained with these combinations is 56.68%. The five AUROC curves of these results can be found in appendix A-D.

TABLE III
F1 SCORE ON THE TEST SET OF EACH COMBINATION OF REPRESENTATIONS AND OUTLIER DETECTION METHODS. BOLD RESULTS DENOTE THE BEST REPRESENTATION SCORE AND UNDERLINED RESULTS DENOTE THE BEST OUTLIER DETECTION METHOD.

F1 score	Size	Cos_Sim	kNN - Exact	kNN - Mean	kNN - H mean	DBSCAN	Isolation Forest	OneClass SVM
Bow	341	**49.72%**	36.38%	36.38%	36.38%	**49.00%**	29.82%	34.12%
TF-IDF	341	40.22%	41.84%	44.49%	43.47%	41.26%	27.30%	**45.37%**
LDA	5	36.03%	26.13%	35.01%	32.94%	35.65%	33.49%	27.64%
	10	40.66%	42.02%	42.02%	41.69%	34.80%	38.40%	34.77%
	20	40.83%	46.33%	46.33%	46.33%	42.82%	**50.35%**	39.86%
Skipgram Sum	5	41.33%	**56.68%**	34.10%	34.10%	46.68%	31.10%	30.61%
	10	48.69%	34.24%	33.88%	32.79%	46.15%	31.75%	31.33%
	20	43.70%	35.01%	35.16%	34.49%	45.37%	32.42%	32.06%
Skipgram Mean	5	41.33%	41.81%	**56.45%**	**55.34%**	40.21%	45.34%	39.80%
	10	48.69%	31.31%	43.01%	43.42%	44.17%	45.25%	36.35%
	20	43.70%	37.54%	44.72%	44.44%	41.53%	38.46%	39.61%

B. Discussion

The results obtained can be explained by different factors. First, even though the dataset used for the experiments has been generated on a recent operating system, it may be too small and simple to benefit from complex models such as neural networks. System calls are only represented by their label, and it is challenging to apply anomaly detection supervised tasks on it. Second, we only used off-the-shelf methods to detect outliers data. As we said earlier, we could manually analyze the sequences of system calls to find unknown bugs or anomalies. However, this practice requires significant work. We could also try tailor-made methods, designed especially for the tasks. However, we do not answer this paper's work objective. Consequently, the only way to determine if we can obtain better results following our line of work is to keep testing other methods and try a new dataset.

V. CONCLUSION

The answer to this internship objective is yes: machine learning models are able to detect anomalies within traces. However, there is a prerequisite: the data must be represented in a fixed-size object that most machine learning and deep learning algorithms can process. In this internship, we mainly used vectors and matrices as they are prevalent in artificial intelligence and very easy to manipulate. This work showed that NLP methods work pretty well on system call sequences.

Further work would be to test other representations and methods to see if better results are achieved. For the representations, we could implement "Distributed Representations of Sentences and Documents" by Le and Mikolov [11], which is an unsupervised algorithm also called Doc2Vec that learns fixed-length feature representations of variable-length pieces of texts. Additionally, we could perform "Sequence to Sequence with Neural Networks" or Seq-2-Seq by Sutskever et al. [12], which presents an approach to map sequence to sequence. As for outlier detection methods, we could try Local Outlier Factor (LOF) proposed in 2000 by Breunig et al. [13] which shares the same concept as DBSCAN.

REFERENCES

[1] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=944937

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 3111–3119.

[4] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1990. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf

[5] D. P. Kingma and J. 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. [Online]. Available: http://arxiv.org/abs/1412.6980

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[7] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, 2014.

[8] D. Bacon, "File system measurements and their application to the design of efficient operation logging algorithms," in *[1991] Proceedings Tenth Symposium on Reliable Distributed Systems*, 1991, pp. 21–30.

[9] J. Wang and Y. Hu, "Profs-performance-oriented data reorganization for log-structured file system on multi-zone disks," in *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2001, pp. 285–292.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[11] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014. [Online]. Available: http://arxiv.org/abs/1405.4053

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[13] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *PROCEEDINGS OF THE 2000 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 2000, pp. 93–104.

APPENDIX A
EXHAUSTIVE RESULTS

A. Accuracy

TABLE IV
ACCURACY OF THE TEST SET OF EACH COMBINATION OF REPRESENTATIONS AND OUTLIER DETECTION METHODS. BOLD RESULTS DENOTE THE BEST REPRESENTATION AND UNDERLINED RESULTS DENOTE THE BEST OUTLIER DETECTION METHOD.

ACCURACY	Size	Cos_Sim	kNN - Exact	kNN - Mean	kNN - H mean	DBSCAN	Isolation Forest	OneClass SVM
Bow	341	**82.33%**	74.52%	74.52%	74.52%	73.74%	48.37%	71.87%
TF-IDF	341	67.32%	81.30%	76.95%	75.10%	**76.64%**	41.28%	77.42%
LDA	5	64.53%	75.86%	68.18%	63.97%	73.10%	65.20%	65.22%
	10	70.92%	74.66%	74.66%	72.20%	69.89%	81.55%	**84.40%**
	20	75.33%	78.15%	78.15%	78.15%	70.56%	81.94%	71.95%
Skipgram Sum	5	79.79%	**86.06%**	70.33%	70.33%	70.22%	69.83%	68.74%
	10	80.83%	70.83%	69.38%	67.51%	68.74%	61.60%	65.00%
	20	75.55%	71.92%	71.90%	70.95%	67.26%	68.24%	67.12%
Skipgram Mean	5	79.79%	85.32%	**85.96%**	**85.18%**	70.95%	73.82%	83.45%
	10	80.83%	82.00%	71.45%	74.69%	67.54%	**81.97%**	74.64%
	20	75.55%	81.61%	77.92%	76.84%	72.73%	68.74%	68.77%

Based on this table, the combination of the representation and the outlier detection method that gives the best score is Skipgram Sum 5 and kNN Exact. The best accuracy score is 86.06%.

B. Precision

TABLE V
PRECISION OF THE TEST SET OF EACH COMBINATION OF REPRESENTATIONS AND OUTLIER DETECTION METHODS. BOLD RESULTS DENOTE THE BEST REPRESENTATION AND UNDERLINED RESULTS DENOTE THE BEST OUTLIER DETECTION METHOD.

PRECISION	Size	Cos_Sim	kNN - Exact	kNN - Mean	kNN - H mean	DBSCAN	Isolation Forest	OneClass SVM
Bow	341	**42.47%**	28.59%	28.59%	28.59%	**34.16%**	18.59%	25.89%
TF-IDF	341	27.42%	38.25%	34.27%	32.48%	32.56%	16.65%	35.04%
LDA	5	24.44%	23.57%	24.92%	22.59%	27.36%	23.21%	19.83%
	10	28.93%	31.51%	31.51%	30.02%	25.42%	37.39%	**44.48%**
	20	31.38%	36.07%	36.07%	36.07%	29.86%	**42.00%**	28.98%
Skipgram Sum	5	35.81%	**51.85%**	25.21%	25.21%	31.58%	23.30%	22.62%
	10	39.90%	25.49%	24.71%	23.47%	30.81%	21.42%	21.93%
	20	32.88%	26.41%	26.48%	25.68%	29.97%	23.49%	22.94%
Skipgram Mean	5	35.81%	49.48%	**51.50%**	**49.33%**	28.71%	32.58%	42.33%
	10	39.90%	35.25%	30.32%	32.19%	29.47%	40.58%	30.14%
	20	32.88%	37.15%	35.20%	34.16%	30.20%	26.96%	27.57%

Based on this table, the combination of the representation and the outlier detection method that gives the best score is Skipgram Sum 5 and kNN Exact. The best accuracy score is 51.85%.

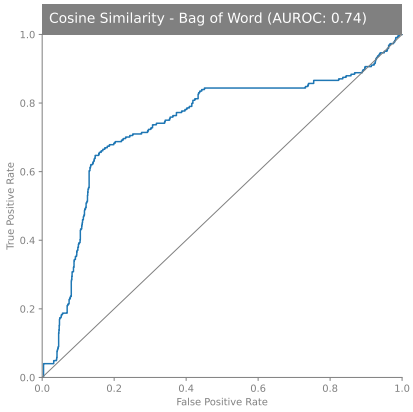
C. Recall

TABLE VI
RECALL OF THE TEST SET OF EACH COMBINATION OF REPRESENTATIONS AND OUTLIER DETECTION METHODS. BOLD RESULTS DENOTE THE BEST REPRESENTATION AND UNDERLINED RESULTS DENOTE THE BEST OUTLIER DETECTION METHOD.

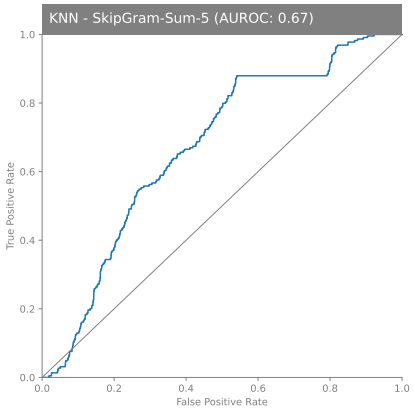
RECALL	Size	Cos_Sim	*k*NN - Exact	*k*NN - Mean	*k*NN - H mean	DBSCAN	Isolation Forest	OneClass SVM
Bow	341	59.96%	50.00%	50.00%	50.00%	86.59%	75.29%	50.00%
TF-IDF	341	**75.48%**	46.17%	63.41%	65.71%	56.32%	**75.67%**	64.37%
LDA	5	68.58%	29.31%	58.81%	60.73%	51.15%	60.15%	45.59%
	10	68.39%	63.03%	63.03%	**68.20%**	55.17%	39.46%	28.54%
	20	58.43%	**64.75%**	**64.75%**	64.75%	75.67%	62.84%	63.79%
Skipgram Sum	5	48.85%	62.50%	52.68%	52.68%	89.46%	46.74%	47.32%
	10	62.45%	52.11%	53.83%	54.41%	91.95%	61.30%	54.79%
	20	65.13%	51.92%	52.30%	52.49%	**93.30%**	52.30%	53.26%
Skipgram Mean	5	48.85%	36.21%	62.45%	63.03%	67.05%	74.52%	37.55%
	10	62.45%	28.16%	73.95%	66.67%	88.12%	51.15%	45.79%
	20	65.13%	37.93%	61.30%	63.60%	66.48%	67.05%	**70.31%**

Based on this table, the combination of the representation and the outlier detection method that gives the best score is Skipgram Sum 20 and DBSCAN. The best accuracy score is 93.30%.

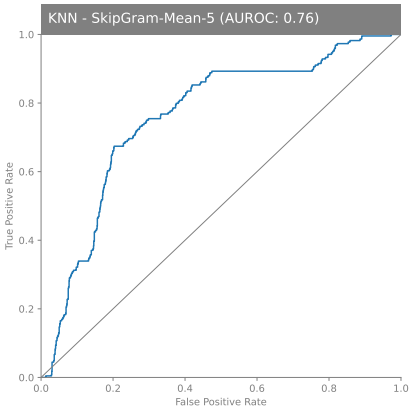
D. Receiver Operating Characteristic



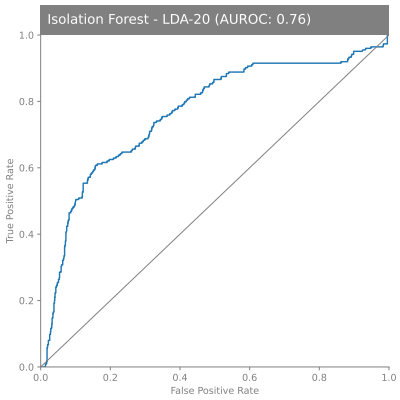
(a) Cosine Similarity - Bag of Word



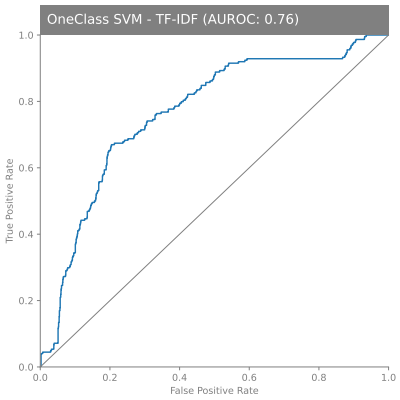
(b) *k*NN - Skipgram Sum 5



(c) *k*NN - Skipgram Mean 5



(a) Isolation Forest - LDA 20



(b) One Class SVM - TF-IDF