EXPLORATION OF NLP-BASED FEATURE EXTRACTION TECHNIQUES FOR SECURITY ANALYSIS AND ANOMALY DETECTION OF SERVICE LOGS

by

Egil Karlsen

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2023

Table of Contents

List of Tables

List of Figures

Abstract

The goal of this research is to provide security and machine learning (ML) practitioners with deeper insight when selecting features and algorithms for unsupervised log analysis. This thesis explores the effect of traditional vector space model and state-of-the-art transformer based natural language processing (NLP) language models towards anomaly detection. Four unsupervised learning algorithms are applied on four service log files using syntactic and semantic feature extraction techniques. This research also explores the use of five different deep learning language models and their impact on the performance in anomaly detection via semantic feature extraction. The results indicate that semantic feature extraction using transformer based language models performs better than the traditional vector space model from the lens of security analysis and anomaly detection.

Acknowledgements

First and foremost, I would like to thank both my co-supervisors Dr. Nur Zincir-Heywood, and Dr. Xiao Luo whose support and guidance was paramount to the completion of this work. I would also like to thank Dr. Jeff Schwartzentruber, whose continued support also made this work possible. Finally, I would like to thank Dr. Vlado Keselj for their timely review and constructive feedback on this work.

Chapter 1

Introduction

Modern day networks are expansive and highly active, particularly in the case of enterprise infrastructure, where applications may be servicing hundreds of thousands of users simultaneously. In most modern implementations, network activity is monitored via log sets, which include relevant details of network events and the timestamps at which the events occurred. This standard pratice results in the generation of many different historical log sets for each system in the network, where log data may take many different forms. These log sets are typically aggregated in a central system (e.g. SIEM) for log analysis and subsequent identification of anomalies. There exist several different approaches for log analysis, from writing regex parsers for specific log lines for analyzing the logs based on specific queries to supervised machine learning based approaches for identifying outliers. NLP techniques have come into the spotlight as an alternative method for log analysis, due to its inherent benfits of increased scalability and generalizability for feature extraction in log analysis when compared to traditional approaches [12, 6].

Evaluation of these log sets through unsupervised machine learning algorithms are valuable in practice due to the difficulty in obtaining labelled or ground truth data most of the time, such as identifing malicious vs. benign network traffic. Research in the application of NLP techniques by Boffa et al. for feature extraction in log analysis have identified the use of Word2Vec and TF-IDF as effective methods of deriving vector space representations of aggregated security data from exposed honeypots [6]. In addition, Copstein et al. explores the use of TF-IDF in combination with unsupervised machine learning algorithms on publicly available network traffic and application log data, identifying it as a valid strategy for anomaly detection [12, 7]. However, exploring the efficacy of NLP-based approaches is relatively recent and have not been fully evaluated. Thus, in this thesis the research objective is to explore the use of both syntactic and semantic feature extraction techniques on four different

security datasets in combination with four unsupervised machine learning algorithms as an anomaly detection solution. To achieve this, this study expands on previous work by utilizing a syntactic extraction technique proposed in [7] based on TF-IDF for security data. In addition to this syntactic approach, this study also extracts semantic features by utilizing different Large Language Transformer models based on the same log data [8]. In doing so, the objective is to explore whether one can identify techniques that are not specific to certain queries or log lines (records) but could be applied to different types of application logs that have different characteristics. Hence, the research contributions of this thesis include:

- Exploring both syntactic and semantic features for application log representation using NLP-based approaches;
- Evaluating four unsupervised learning techniques with syntactic and semantic features on four different types of application logs;
- Interpreting the results through data visualization and case study;
- Evaluate five different large language models for semantic feature extraction for log analysis;
- Compare and contrast the benefits of fine-tuning in large language models for the task of feature extraction in log analysis.

The rest of the thesis is organized as follows: Chapter 2 reviews the literature in this field. Chapter 3 introduces the proposed approach and discusses the datasets, data pre-processing, feature selection, and unsupervised learning models used in this thesis. Chapter 4 presents the evaluations and the results obtained. Finally, conclusions are drawn and future work is discussed in Chapter 5.

Chapter 2

Literature Review

Log analysis is an important area of cybersecurity, where several works studied different log abstraction techniques for different types of log data analysis. Most of the time, the output of an abstraction technique is used as a feature set for intrusion detection (supervised learning) or anomaly detection (unsupervised learning) purposes for data analysis [7, 12]. The generated feature sets were proposed to reduce the ambiguity of malicious behavior visible in the log data. The success of those features was evaluated based on performance metrics derived through the application of learning algorithms.

Nguyen et al. [28] suggest a supervised ensemble learning system which fuses the outputs of different supervised machine learning models together to optimize the classification of abnormal security events. This framework is tested on both the ECML/PKDD 2007 dataset, and the HTTP CSIC 2010 dataset using hand picked features, and syntactic characteristics of the log line. They identify the importance of adapting intrusion detection systems (IDSs) in heterogeneous and adversarial network environments in testing this adaptive intrusion detection system. The authors proposed A-IDS achieves 10% higher accuracy than the best of four different baseline supervised machine learning by combining their outputs using an online learning framework. The unsupervised machine learning algorithms that make up this A-IDS are decision stump, Naive Bayes, RBF Network, and Bayesian Network. A major contribution of this work is a novel expanded framework for fusing the outputs of the four IDSs to increase accuracy and reliability. The A-IDS scores a 90.98% accuracy in the CSIC dataset, and 92.56% in the PKDD/ECMl dataset, albeit at much higher computational cost to perform the output fusion of the underlying models.

Bhatnagar et al. [4] benchmark the performance of four different supervised machine learning algorithms for intrusion detection. The proposed solution extracts and learns features from the raw log data using a trained classical neural network and

deep belief network. These models enable key features to be extracted from both the ECML/PKDD 2007 and HTTP CSIC 2010 dataset, to then have the top five highest entropy features be selected for the featureset. In addition a bag-of-words vector space is created to supplement these features. This hybrid featureset is then used to train four different supervised classifiers for intrusion detection. The four different supervised classifiers are MLP classifier, Multinomial classifier, decision tree, and random forest. The classifiers all score above 99% accuracy in identification of attacks across both datasets.

Vartouni et al. explored the use of a novel feature extraction method for intrusion detection using deep belief networks and parallel feature fusion techniques. They explore the synthetic feature extraction and feature fusion is performed on top of the bi-gram representation of the logsets, alongside typical dimensionality reduction algorithm, like FICA, PCA, and KPCA. They evaluate the benefits of dimensionality reduction, synthetic feature fusion, and baseline n-gram representation by training unsupervised anomaly detection algorithms such as Isolation forest, Eliptic envelope, and one class SVM and comparing the outlier detection performance using accuracy, and F1-score. Benchmarking is performed using both the ECML/PKDD 2007 dataset, and the HTTP CSIC datasets. The results indicate that there is deep learning neural networks, particularly fusion models can provide meaningful benefits to feature extraction in the task of anomaly detection. The most remarkable results show the feature fusion method at a 85.35 F1-Score in the CSIC dataset for isolation forest, and 84.93 F1-Score using elliptic envelope on the PKDD dataset.

Copstein et al. [7] explored the use of syntactic features based on the TF-IDF technique for intrusion detection in application log files. TF-IDF is a simple natural language processing technique which enables ranking of terms based on their uniqueness in the context of the entire dataset; they suggest this feature set as a method of extracting the form of log line data and in doing so syntactical attributes which are important for anomaly detection. This approach was evaluated using unsupervised clustering algorithms K-Means, DBScan, and EM clustering. Three datasets were used in this research, ISOT-CID a cloud intrusion dataset, ECML/PKDD 2007 dataset and an Indonesian Apache Access security set. The syntactic approach is compared and contrasted against more traditional log abstraction techniques which are

inflexible, indicating through benchmarking that a syntactical approach does effectively and consistently disambiguate malicious behavior in traffic log data of different forms. They record their highest performance using K-Means clustering with accuracy scores between 60-70% using K-Means clustering for anomaly detection across the three datasets, with arbitrary selection of K clusters.

In more recent works, Nam et al. [27] explored the use of BERT for log analysis in VM failure prediction. They propose a model in which BERT is used to extract the sentence embedding representation of each log and feed that into a pre-trained convolutional neural network to predict failure within a 2 to 30 minute window. They achieved a 0.74 F1 score on an in-house OpenStack VM system log test and training set. A notable beneift of this BERT-CNN model was that the embedding representation enabled the CNN to predict VM failure in instances where the problem was not observed in the training set.

Ott et al. [29] evaluates the use of language models BERT, GPT-2 and XL-Net for anomaly detection in cloud system log data. They accomplish this by leveraging different base language models for extracting key components of log data, and enabling transfer learning by approximating the similarity of key log events across different datasets with slightly altered sequences and semantic meaning. All log data was pre-processed into a log template representation using Drain prior to semantic feature extraction using the language models. They train two neural networks with a regression and classification learning objective in which they prioritize mean squared error for the anomaly detection component of this proposed solution, which takes the sentence embedding representations as input from the language models. They evaluate robustness of the use of the three different language models by gradually altering the log input data in both semantic representation and sequence and evaluating the change in model performance. The results for BERT and GPT-2 indicate they are less susceptible to semantic and sequence changes in the regression based solution but GPT-2 underperforms in the classification-based model. Additionally BERT benefits from better transfer learning performance in the proposed solution indicating it is a good candidate for anomaly detection in an evolving system.

Zhang et al. [40] explore the modification of the BERT base model to perform weighted part-of-speech tagging on log templates produced by Drain and improve

software specific classification for log events across five different system log datasets. They modify BERT to accept a new embedding representation in place of segment embeddings which is the summation of the token, position and PoS weight embedding which is gathered from a predefined table where tags are given relevant weights based on expert advice for log classification. PoSBERt is compared to base BERT in log software classification and the added weight PoS embedding and achieves a 0.3-5% across four of the 5 datasets.

Table 2.1: Literature Overview

Paper	Unsupervised	Syntactic	Semantic	Application Log
[28]	NO	NO	NO	YES
[4]	NO	YES	NO	YES
[26]	YES	YES	NO	YES
[7]	YES	YES	NO	YES
[16]	YES	NO	NO	YES
[21]	YES	NO	NO	YES
[27]	NO	NO	YES	NO
[29]	NO	NO	YES	NO
[40]	NO	NO	YES	NO
This Thesis	YES	YES	YES	YES

2.1 Summary

Table 2.1 presents an overview of the literature summarized in this chapter. The literature in feature extraction and selection for log analysis and anomaly detection leans heavily on supervised machine learning approaches and their supporting solutions to feature extraction. The inflexibility of these implementations as they explore only one type of log data and often extract specific information provides a research gap on which to explore new approaches to feature extraction and unsupervised machine learning for more realistic evaluation and benchmarking efforts. Furthermore, there is even less research on the semantic feature extraction side, in terms of evaluating different language models on application log analysis for security purposes. Different from the research in the literature, the goal in this thesis is to explore the effect of syntactic and semantic features using unsupervised learning techniques for designing and developing anomaly detection strategies on application log files. In doing so, the aim is to benchmark and explore how different popular language models perform when tasked with extracting meaningful sentence embedding representations

from security log data. Thus, the thesis extends the approach of Copstein et al. [7] to explore the effect of syntactic and semantic features using unsupervised learning techniques for designing and developing anomaly detection strategies on application log files. Moreover, the focus is specifically on application log files, namely web log files. To the best of my knowledge, this is the first work benchmarking well known unsupervised learning algorithms with such an approach on four distinctly different and publicly available application log files. This research also aims to explore the use of different language models for semantic feature extraction and how they differ from one another.

Chapter 3

Methodology

This chapter will explore the different datasets, feature extraction methods, visualization methods, and unsupervised machine learning algorithms utilized in this research. This section explores the components that went into the implementation of an experimental methodology for evaluation of syntactic and semantic feature extraction techniques for security log analysis, and anomaly detection. The figure 3.1 provides an overview of this experimental structure.

3.1 Datasets

This thesis explores the efficacy of a generalized approach to log analysis which leverages the flexibility of feature engineering approaches in syntactic and semantic analysis on four unique application log files. Since a primary component of this research is to explore a generalizable approach to application log analysis for security purposes, it is important to employ data from different sources. Therefore, four publicly available datasets are employed in this research. Each one represents a different real life system summarized below, Table 3.1 and Table 3.6. It should be noted here that each dataset was split into two partitions: 75% training and 25% testing, using stratified sampling to maintain the original class distributions of the data. For the purposes of this research the training and test sets were reduced to a binary representation of their original classes, that is to say classes associated with malicious behavior were all labelled as ANOMALY and safe activity were labeled as NORMAL. The details of each dataset is described as follows:

Table 3.1: Summary of the datasets

	PKDD		ISOT		AA		CSIC	
Class	#Lines	%	#Lines	%	#Lines	%	#Lines	%
NORMAL	35006	69.85%	704,226	69.88%	29,778	85.71%	36,000	59.01%
ANOMALY	15109	30.15%	303,529	30.12%	4,965	14.29%	25,000	30.99%
Total	50,115	100%	1,007,756	100%	34,743	100%	61,000	100%

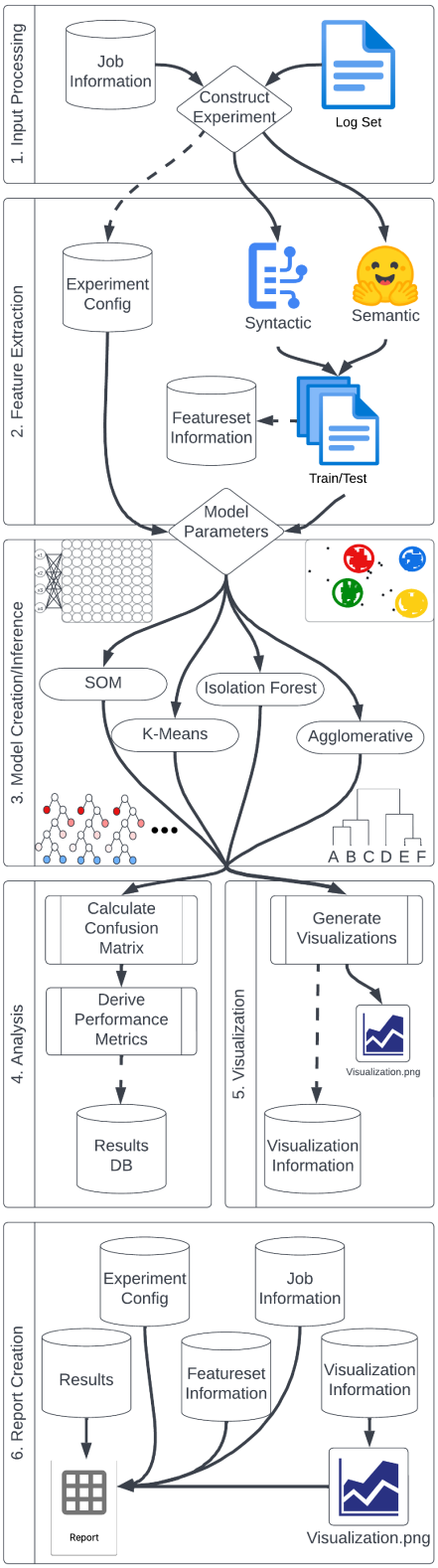

Figure 3.1: Overview of the System Framework

3.1.1 ECML/PKDD Dataset

This dataset was created and published as training/testing data for anomaly detection on web application data for the ECML/PKDD Discovery Challenge 2007 [11]. The dataset contains 50,115 log records which are composed of HTTP packet information distributed across normal and attack behaviours, Table 3.2. This dataset represent a scenario where certain parts of data is anonymized to avoid releasing personally identifiable information. This dataset has the highest number of words among the ones used in this research. Hereafter, this dataset is referred to as PKDD. The features which are present in this dataset as part of the log line representation are as follow:

1. Method
2. URL
3. Content-type
4. Content-language
5. Content-encoding
6. Content-location
7. Content-MD5
8. Content-Type
9. Expires
10. Last-Modified
11. Host
12. Connection
13. Accept
14. Accept-charset
15. Accept-encoding
16. Accept-language
17. Cache-control
18. Source IP
19. Cookie
20. Cookie2
21. Date Time
22. Etag
23. Expect
24. From
25. If-Modified-Since
26. If-Unmodified-Since
27. If-Match
28. If-None-Match
29. If-Range
30. Max-forwards
31. MIME-Version
32. Pragma
33. Proxy-Authorization
34. Authorization
35. Range
36. Referer
37. TE
38. Trailer
39. User Agent
40. UA-CPU
41. UA-Disp
42. UA-OS
43. UA-Color
44. UA-Pixels
45. Via

46. Transfer-Encoding
47. Upgrade
48. Warning
49. X-Forwarded-For
50. X-Serial-Number
51. Content-length
52. Content
53. URL

Table 3.2: ECML/PKDD Dataset

Class	#	%	#	%
C1 - Normal	35,006	69.84%	35,006	69.85%
C2 - Cross-Site Scripting	1,825	3.64%	15,109	30.15%
C3 - SQL Injection	2,274	4.53%	-	-
C4 - LDAP Injection	2,279	4.54%	-	-
C5 - XPATH Injection	2,279	4.54%	-	-
C6 - Path Traversal	2,295	4.57%	-	-
C7 - Command Execution	2,302	4.59%	-	-
C8 - SSI Attacks	1,856	3.70%	-	-
Total	50,115	100%	50,115	100%

3.1.2 CSIC Dataset

This dataset was created in 2010 by the Information Security Institute by the Spanish Research National Council as an updated alternative to the KDD cup dataset from 1999 for the research and development of intrusion detection tools on web application logs [13]. It was produced by simulating an e-commerce site, Table 3.3. The simulated attacks include SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server side include, parameter tampering. This dataset is referred as CSIC, and includes a feature list comparable to that of the PKDD dataset without the anonymization. The features which are present in this dataset as part of the log line representation are as follow:

1. HTTP method
2. User Agent
3. Pragma
4. Cache-control
5. Accept
6. Accept-encoding
7. Accept-charset
8. Language
9. Source IP
10. Cookie
11. Content-type
12. Connection
13. Content-length
14. Content
15. URL

Table 3.3: CSIC Dataset

Class	#	%
Benign	36,000	59.01%
Malicious	25,000	30.99%
Total	61,000	100%

3.1.3 ISOT-CID Dataset

This dataset was created and published by the ISOT lab to be used as training and testing data for intrusion detection on cloud environments in 2018 [35]. The full dataset is composed of 8tb of system logs, calls, traffic data at VM and hypervisor levels and system performance data collected across 10 days. We selected the network traffic data for day-8 because of its relatively smaller size, as shown in Table 3.4. This log file represents the scenario of tcpdump traffic and includes both attack and normal traffic. This is the biggest dataset used in this research. Hereafter, this dataset is referred to as ISOT. The features which are present in this dataset as part of the log line representation are as follow:

1. Transport Layer Protocol
2. Source IP
3. Destination IP
4. Source Port
5. Destination Port
6. Length of the packet
7. Promiscuous mode FLAG
8. Packet timestamp
9. Sequence number
10. TCP flag

Table 3.4: ISOT-CID Day-2 Dataset

Class	#	%
Benign	7,157,658	61.90%
Malicious	4,404,874	38.10%
Total	11,562,532	100%

3.1.4 Apache Access Dataset

This dataset was created and published by Indramayu State Polytechnic University for training and testing of intrusion detection on web server access logs in 2022 [15]. The dataset is composed of 37,693 log records including attack and normal queries, Table 3.5. This dataset represents a regular application log file, no anonymization employed. Hereafter, this dataset is referred to as AA. The features which are present in this dataset as part of the log line representation are as follow:

1. Source IP
2. Date Time
3. Timezone
4. URL
5. Method
6. Status Code
7. Response Size
8. User Agent
9. Country

Table 3.5: Apache Access Dataset

Class	#	%	#	%
AMA - Safe	29,778	79%	29,778	85.71%
BAH - Attack	4,965	13.17%	4,965	14.29%
DIC - Suspicious	2,950	7.83%	-	
Total	37,693	100%	34,743	100%

Table 3.6: NLP Characteristics of the datasets

Dataset	Word Length	Word Count	Character Count
PKDD	11.49	92.1	1124.25
Standard Deviation	1.67	12.32	168.01
ISOT	6.35	10.47	75.96
Standard Deviation	0.21	0.50	3.28
AA	12.22	16.85	222.13
Standard Deviation	2.87	1.83	46.07
CSIC	15.94	27.55	467.02
Standard Deviation	3.50	0.90	104.13

3.1.5 Harvard Web Server Dataset Description

This dataset is composed of 10,365,152 Apache access log lines created from an Iranian e-commerce website zanbil.ir and is publicly available on Harvard dataverse [39]. This is 3.5 GB of raw Apache access logs gathered over a 4 day period between January

22 2019 and January 26 2019. This dataset is only used to finetune the selected transformer language models.

3.2 Feature Extraction

This research explores two different NLP techniques for feature extraction with no a priori information from the datasets. That is to say these NLP techniques are applied without fine-tuning or specialization on the log formats seen in the data. The two techniques extract information at different levels of NLP, one for syntactic and one for semantic. The semantic technique attempts to extract meaning from contextual information in the log, looking at relationships between tokens within the log. While the syntactic approach attempts to interpret the data from its syntax representation, identifying important words from their frequency in the scope of the entire dataset.

3.2.1 Syntactic: TF-IDF based Feature Extraction

In the study by Copstein et al. TF-IDF, a lightweight syntactical feature extraction technique, is developed for security log analysis and evaluated against the traditional and computationally expensive log abstraction methods [1]. Thus given a log line, d with tokens ts, and a log set, D, composed of N, a TF-IDF vector representation can be calculated by first deriving the token frequency using Eq. 3.1. Then deriving the inverse log line frequency of the token using Eq. 3.2. Finally, the product of Eq. 3.1 and 3.2 provides the TF-IDF for one token t in the log line, shown as Eq. 3.3 [1].

$$tf(t,d) = freq(t,d) \tag{3.1}$$

$$idf(t,D) = log(\frac{N+1}{count(k \in D : t \in k)+1}) + 1 \tag{3.2}$$

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D) \tag{3.3}$$

Since the TF-IDF representation of each log line can be verbose and vary greatly for some security log types, in the interest of conformity, model training times and generalization, the average, minimum and maximum TF-IDF values are calculated from each log line as the three TF-IDF features in the short syntactic feature set [7].

TF-IDF is calculated for each log line for every term including numbers since they are of significant importance to log lines. Other than these three features derived from TF-IDF, four additional syntactic features are produced from each log line to build the syntactic representation of it. The four additional syntactic feature are calculated as follows:

- Alphanumeric Ratio: Ratio of alphanumeric characters over the total number of characters in the log line [7]
- Average Character: Average of the numeric ASCII representation of each character in the log line
- Character Count: Total number of characters in the log line
- Word Count: Total number of words (character sequences separated by whitespace) in the log line

TF-IDF is a Natural Language Processing technique in which sentences are translate into vector value representation through analysis of individual terms and their frequency relative to the full logset. This technique is predicated on the fact that the document (log line) belongs to a larger document model (logset) on which the components of TF-IDF can be calculated for each term.

Thus given a document (log line), d, and a document model (log set), D, composed of N documents and a term, t; a TF-IDF vector representation can be calculated by first deriving the term frequency using the equation 3.1. Then deriving the inverse document frequency of the term using the equation 3.2. Finally, the product of the two provides the TF-IDF for one term in that document seen in equation 3.3.

Evaluation of TF-IDF provides an understanding that as the frequency of a term in its given document increases and its frequency in the context of the rest of the document model decreases, TF-IDF for that term will increase — and vice versa. Thus, a term with high TF-IDF score is uncommon and by extension an outlier.

The TF-IDF vector representation of each log line is generated by turning the log set into a document model, composed of each log line as documents and then computing the TF-IDF value for each word present. This is all performed exclusively on the log line data. Since the TF-IDF representation of each log line can be verbose

Figure 3.2: TF-IDF calculations of sample ISOT log lines

and vary greatly on the application log types, in the interest of model training times and generalization the average, minimum and maximum TF-IDF values are selected from each log line as the three TF-IDF features in the short syntactic feature set employed in this thesis.

3.2.2 Semantic: LLM based Feature Extraction

For the semantic approach this study utilizes the state-of-the-art Bidirectional Encoder Representation from Transformers model (BERT) to derive the embedding representation of each log line [8]. Originally released in 2018, the BERT model utilizes the transformer architecture to learn the contextual relationship between words in text. The use of BERT on security log data was previously explored in a study by Guo et al. [14] where they use the BERTs ability to perform next sentence prediction and fill in missing data in order to identify anomalous events in log information. This study primarily focused on system logs.

3.2.2.1 BERT

The BERT model stands for bidirectional encoder representations from transformers, this is an open source machine learning framework for natural language processing produced by Google in 2018 [8]. The purpose of which is to enable computers to better understand ambiguous text, by looking at the context and structure of the language. This understanding of context is enabled by the unique bidirectional property of BERT models, which allows the model to process text not only left-to-right or right-to-left but both simultaneously.

The BERT model is based on the transformer architecture which enables neural networks to be trained without fixed sequences. Transformer architectures were introduced by Google as well in 2017, and the benefits of the unordered training data requirements for the transformer architecture provided the opportunity to train on a massive corpus of data [38]. This was how the original BERT model was pre-trained using a 16GB corpus of text from Wikipedia and the brown corpus using both masked language modeling and next sentence prediction.

3.2.2.1.1 Model Architecture

The original BERT model was trained on a vast dataset of over 3.3 Billion words gathered from various sources including Wikipedia and BooksCorpus on the principle that by pre-training an unsupervised multilayer transformer model on written text from a variety of sources where words will have different contextual meanings the model will be better equipped to interpret the contextual meaning of a word [8]. The resulting model has a transformer architecture which utilizes multilayer transformer encoders to derive embedding information using a variety of different text features. The base BERT model has 12 transformer layers each with 768 hidden states, which essentially represent the output of each encoding layer. Prior to going through the different encoding layers of the model the input text is transformed into its embedding representation which describes different characteristics of the sentence. These are the segment embedding, positional embedding, token embedding. These three embedding representations are summed up to provide the input for the transformer encoding layers. The hidden state output of each encoding layer can then be used as semantic information. The hidden state that is produced by the final layer of the BERT model can then be used as the semantic representation of the input text.

3.2.2.1.2 Sentence Embedding Extraction

Sentence or sequence level analysis using BERT and BERT like models is accomplished by extracting the hidden states or token embedding of the final layer of the BERT model and performing mean pooling on that token level representation. This effectively compresses the token embeddings into a fixed length sequence which can represent the meaning of the input sequence [31]. Therefore, in order to successfully extract the sentence embedding from the language models utilized in this thesis a mean pooling layer is added following the final layer of each model. Please refer to Figure 4.2 for a visualization of the appropriate model architecture to extract sentence embeddings.

3.2.2.1.3 Wordpiece Tokenization

BERT processes text by performing tokenization with the WordPiece algorithm, where an underlying vocabulary is extracted from the pre-trained corpus which includes every character present in the text as well as characters which are preceded by other characters [8]. For example, the word "giraffe" would be split into the following "g" ""##i" "##r" "##a" "##f" "##e" where every character inside the word is preceded by a prefix of "##". Then in order to expand the vocabulary, merge rules are extracted from the corpus to define longer sequences of tokens which are prevalent in the text.

3.2.2.1.4 Masked Language Model

One mechanism used to pre-train the BERT model is the attention mechanism or masked LM which masks 15% of keywords in the text prior to being fed into the model so as to enable the model to train in accurately predicting the masked word based on contextual information gathered from the rest of the text [8]. This masked LM mechanism is predicated on the implementation of a classification layer on top of the encoding layer which attempts to assign the correct word to masked words, as well as transformation of the embedding matrix output into their vocabulary representation and finally computing the probability for each word using the softmax layer. This explanation can be seen in Figure 4.2.

3.2.2.1.5 Next Sentence Prediction

The other mechanism utilized in pre-training the BERT model is next sentence prediction [8]. This mechanism utilizes the different embedding representations, especially the segment embeddings to have the model predict the consequent sentence of a given sentence. During training 50% of the inputs are paired with their subsequent sentence in their original documents, another 50% is paired with a random sentence that is not its original subsequent sentence. The goal here in next sentence prediction is that the sentences paired with an incorrect subsequent sentence are correctly identified as not pairs. This mechanism is implemented at the softMax layer where the model will produce a confidence interval for determining if the two sentences separated as

segments are subsequent sentences. While the BERT model produced by Google provided an entirely new method of approaching NLP tasks, there have since been many adaptations made on the BERT framework to improve its performance in tasks in different ways. This study explores some of the unique transformer style language models that have been released since 2018 for the task of semantic analysis of security log data.

3.2.2.1.6 Fine-tuning

BERT model fine-tuning is performed in much the same way as pre-training. Masked language modeling is used to adapt the language model to a specific domain or task during fine-tuning [8]. In the context of this research each model was fine-tuned without freezing, which is the practice of ensuring some layers of the transformer models are not updated during fine-tuning. Every layer in the context of this study was impacted during fine-tuning.

3.2.2.2 Roberta

The RoBERTa model is a BERT-style language model produced by the Facebook AI division in 2019 [23]. This model differs from the original BERT model in a number of ways and stands for Robustly Optimized BERT Approach. The model authors at Facebook AI made key decisions to improve on the original BERT model architecture and outperform it in several NLP tasks like text classification, and translation. The following is a list of unique characteristic and key design choices for the RoBERTa model as compared to the original BERT model.

1. Pre-training data: RoBERTa was trained on 160GB of text which includes the 16GB corpus used for BERT and the rest from web and news related text sources.

2. Dropped NSP: RoBERTa has had the next sentence prediction component of the architecture stripped from it since research suggested it did not contribute meaningfully to the overall performance of the BERT model.

3. Difference in Pre-Training Configuration: RoBERTa had much larger batch sizes with singificantly fewer steps. Larger batch sizes and fewer steps led to improved

performance in terms of perplexity, and greater oppurtunity for parallelization and faster pre-training.

4. Dynamic Masking: RoBERTa dynamically changes the location of the masked token in the input sequence during masked language modelling training unlike BERT which utilizes static masking.

5. BPE Tokenization: RoBERTa utilizes a vocabulary of greater than 50,000 tokens as compared to BERT with just over 30,000 as a result of its use of a more comprehensive tokenization algorithm in the form the Byte Pair Encoding algorithm which sacrifices compression for better detail, and emphasis on learning merge set rules for tokens.

3.2.2.3 DistilRoberta

NLP and Machine Learning researchers have not only attempted to adapt the architecture of BERT-style language models for better performance, they have also made advances towards improving the efficiency of those models. A process known as distillation was developed and explored in a study by Sanh et al. in 2019 [32]. This is a process in which large language models are used to train smaller models using a teacher-student compression technique where the aim is to perform knowledge transference and distillation. In the case of language models like BERT or RoBERTa knowledge distillation is performed by training a smaller model to mimic the output of those original models. The student models training loss function is determined by how its output relates to the teachers output, and minimizing the difference on a given pre-training dataset. This process of knowledge distillation has been applied to several language model architectures. The benefits of knowledge distillation is that the number of layers, parameters and pre-training time can be reduced significantly enabling faster inference while replicating the same output as that of a larger model.

The distilRoBERTa model contains only 6 layers, 768 dimensions and 12 heads to to total 82 million parameters which is 12 layers, 768 dimensions and 12 heads for a total of 125 million parameters [32]. The reduction in model size enables double the inference time for DistilRoBERTa.

3.2.2.4 GPT-2

The GPT-2 model is an open source language model produced by OpenAI in 2019, this model differs significantly from the BERT-style models previously outlined [30]. GPT-2 was designed to perform translation, text generation and question answering and has an autoregressive decoder style transformer architecture. Decoder style transformer architecture differs from the encoder style transformer of BERT-style models in that they perform unidirectional input processing and generate each output token individually. GPT-2 is an autoregressive model which aims to predict subsequent tokens using a set of input words, while BERT utilizes autoencoding with its masked language modeling tasks. Autoregressive decoder styler transformer models like GPT-2 are referred to as causal language models. GPT-2 is trained on 40 GB of textual data extracted from web pages, and the original model has 1.5 billion parameters, though smaller models exist and for the purposes of this study a 124 million parameter model was used. Much like the RoBERTa model the GPT-2 model utilizes BPE tokenization, and has a similar size vocabulary to that of RoBERTa with just over 50,000 tokens.

Since the architecture of GPT-2 differs from that of the BERT style models, there are some changes that must be made to the model in order to perform sentiment analysis [30]. Causal language models don't typically need end of sequence tokens to identify the end of input sentences, and so in order to generate fixed length sentence embeddings from the final decoder layer of GPT-2 model, these tokens must be added. Furthermore addition of padding, unknown and beginning tokens is also required in order to map input sequences to a fixed length sentence embedding. After this change the output of the final layer can be seen as the token embedding representation of any input sequence and fed into a mean pooling layer to extract the sentence embedding representation. This is how sentiment analysis can be performed with causal language models.

3.2.2.4.1 Fine-tuning

Causal language models differ in the way in which fine-tuning and pre-training are performed [30]. Rather than attempting to fill in unknown masked words in a sequence, these language models attempt to predict the next word in the sequence

Table 3.7: Model Characteristic of the Different Large Language Models employed

	Model Characteristics		
Language Model	Word Embedding Space	Parameters	Training Data Size
Bert-base-cased	768	110M	16 GB
RoBERTa	768	123M	160 GB
DistilRoBERTa-base	768	82M	40 GB
GPT-2	768	124M	40 GB
GPT-NEO	2048	1.3B	800 GB

given a stub taken from the training corpus. The training goal here is to correctly guess the subsequent words/tokens.

3.2.2.5 GPT-Neo

The GPT-NEO model is a Eleuther AI replication of the closed source GPT-3 model produced by Open AI and is an autoregressive decoder style language model [5]. This model is also a casual language model and utilizes a slightly modified autoregressive decoder style architecture to that of GPT-2. GPT-NEO differs from GPT-2 in that it utilizes local attention between every other layer, and was trained on the 825 GB dataset the pile. The pile training is composed of a wide variety of different text sources including books, github repositories, webpages, chat logs, as well as research papers from a number of STEM fields. The GPT-NEO model comes in a variety of different model sizes as well ranging from 2.7 billion parameters to 125 Million. For the purposes of this research the 1.3 BIllion parameter model was utilized.

This causal language model also requires the addition of end of sentence, beginning of sentence, padding and unknown tokens in order to generate fixed length sentence embeddings from the input sequences [5].

3.3 Unsupervised Learning Techniques

In this research, the unsupervised learning techniques used include Self Organizing Maps, K-means, Isolation Forest, and Agglomerative Hierarchical clustering. They are chosen based on the works reported in the literature, Section 2. Further details regarding unsupervised learning and clustering can be found in Alpaydin et al. [2].

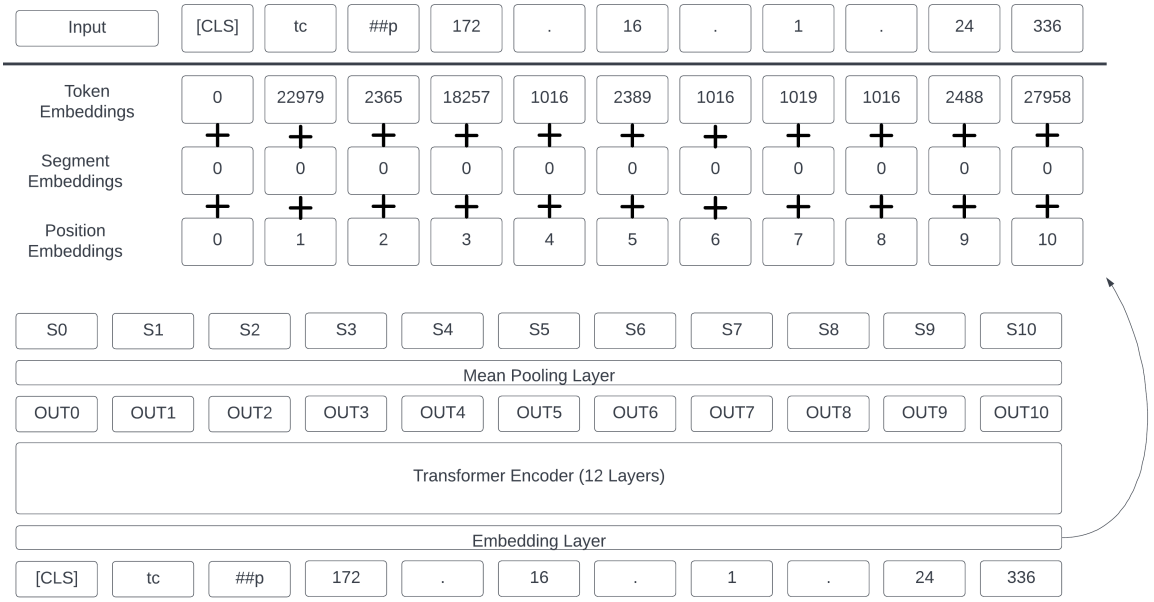

Figure 3.3: An example of BERT Embedding on ISOT dataset

3.3.1 K-Means

The K-Means algorithm is a well-established iterative clustering algorithm [3, 24]. The algorithm attempts to cluster/group data points, and is widely used in fields such as computer vision, data science, and cybersecurity. The underlying principle for K-Means is to ensure that similar data points are clustered together and dissimilar ones are separated, and this is accomplished by minimizing the sum of distances between each data point and their corresponding cluster centroid. The following is a walk through of how the K-Means clustering algorithm works.

1. Initialization: In this first step the algorithm starts with a k randomly selected cluster centroids, where k represents the number of clusters

2. Assignment: The Euclidean distance is then calculated between every data point and every centroid in order to find which data points belong to which cluster centroids. The formula for Euclidean distance can be seen in equation 3.4. where p_i and q_i are the i-th coordinates of points p and q, respectively, and d is the number of features. The symbol $\sum$ represents the summation over all features.

3. Update: Once every data point has been assigned to a cluster centroid, the

position of each cluster centroid is updated too the mean of all data point belonging to its cluster.

4. Re-assignment: After updating the centroids each data point must once again have its closest centroid computed, and assigned to.

5. Repeat: The Update, and Re-assignment steps are repeated until there is little to no change in the centroid positions anymore, or until a user specified number of iterations has been reached. This is where convergence of the model can be reached, though it may not always be the global optimum.

$$d(p, q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{3.4}$$

The outcome of this algorithm is a set of K cluster centroids with data points assigned to them. The low computational cost of computing distance calculations and updating the positions of the centroids makes k-means a highly effective unsupervised machine learning algorithm for performing clustering larger datasets.

3.3.2 Agglomerative

The agglomerative clustering algorithm is a bottom-up hierarchical clustering algorithm [41], [10]. Hierarchical algorithms perform clustering by starting with either one or n clusters, and merging or splitting to cluster data points, since agglomerative is bottom-up it performs hierarchical clustering by starting with n clusters and merging until a stopping criterion is met. The following is a step-by-step description of how the agglomerative algorithm performs clustering [41].

1. Initialization: Each data point is assigned to a cluster

2. Calculate Distance Matrix: The pairwise distance between each cluster is computed in this step. Distance between cluster can be calculated in many different ways in the context of this research ward linkage was used exclusively. This algorithm aims to calculate the distance between two clusters as the sum of squared differences in the clusters variance after the two clusters are combined. See equation 3.5 which computes the ward linkage distance between two clusters A

and B. In this equation $|A|$ $|B|$ are the number of points that have been put in clusters A and B, $\bar{x_A}$ and $\bar{x_B}$ is the cluster points means, and $||\bar{x_A} - \bar{x_B}||_2$ is the euclidean distance between the means. Euclidean distance can be swapped for another distance metric however for the purposes of this reserach only euclidean distance was used.

3. Find Nearest Neighbouring Clusters: Use the distance matrix to determine which two clusters are closest together

4. Merge Nearest Neighbouring Clusters: Merge the nearest neighbouring clusters according to one cluster

5. Update: Re-calculate the distance matrix now that a new cluster has been created from two

6. Repeat: Iterate on the the last three steps until the user defined stopping criterion is met, which could be a fixed number of data points per cluster, number of clusters, and a maximum distance threshold.

$$d_{AB} = \sqrt{\frac{|A||B|}{|A| + |B|}} \cdot ||\bar{x_A} - \bar{x_B}||_2 \tag{3.5}$$

The agglomerative clustering algorithm can be produce a visualization that represents the hierarchy of clusters, called a dendrogram, and this representation can allow the user to cut the model at different heights limiting the number of clusters.

This algorithm is more computationally expensive than the K-Means algorithm but can yeild insight on unique nested structures within data [10].

3.3.3 Isolation Forest

The Isolation Forest algorithm prioritizes outlier/anomaly detection by isolating anomalous data points from normal ones [22]. The algorithm accomplishes this task of identifying anomalies by generating binary trees from the input feature space and partitioning those trees recursively, until each data point is isolated in a partition leaf node of the tree. The following is a step-by-step description of how the isolation forest algorithm works.

1. Pre-selection of Data Point: On every recursive iteration of this algorithm a random subset of data points is selected from the dataset

2. Construct Isolation Tree (iTree): From that randomly selected subset of the data points construct a binary tree that partitions itself by randomly splitting the feature space along different directions. The feature that will be split is randomly selected and is split by selecting a random threshold value between the minimum and maximum range of that feature. The tree is constructed by adding data points to the left or right of the tree based on the splitting criterion. Perform this splitting until each data point is isolated on a partitioned leaf node, or a user-specific maximum tree depth has been reached.

3. Repeat: The first two steps are repeated until a desired number of trees is reached and hopefully the model has converged. Convergence usually occurs within 100 iTrees [22].

4. Anomaly Score: After construction of the forest the anomaly score of each data point can be calculated. This is an aggregated value determined by the average path length to reach the data point in each iTree in the forest. It follows that anomalous data would be much more easily isolated from the rest, therefore they would not have to travel as deep into each iTree in order to find isolation.

5. Contamination: A user defined threshold between (0,0.5] helps to determine whether an anomaly score is indicative of the data point being an outlier/anomaly. This value is called contamination and indicates the percentage of anomalies assumed to be present in the dataset.

The isolation forest algorithm benefits from being able to identify outliers without significant computational costs in high dimensional data [22]. The process of computing each iTree is the most costly step but the identification of anomalous data by how easily it is isolated from the rest of the data holds true, and provides a unique approach to anomaly detection.

3.3.4 Self-Organizing Maps

The Self Organizing Map (SOM) is an artificial neural network based unsupervised learning algorithm [18][19]. SOM produces a lower-dimensional representation of high dimensional data which preserves the pairwise distance relationship of the original data. The goal of SOM learning algorithm is to cause different parts of the neural network to respond similarly to certain input pattern mappings. During mapping, there is a single winning neuron whose weight vector lies closest to the input vector, e.g. Euclidian distance.

That is to say the topology of the original dataset is maintained during dimensionality reduction and in addition eventually more deep insight into the relationship between similar data points is annealed through this approach. This is accomplished by first defining a single layer neural network grid of size $A \times B = C$ where each neuron in this grid has a randomized weight vector. The grid is structured such that each neuron has a neighboring set of neurons described by the connectivity pattern parameter defined on instantiation. It can be assumed for this research that the connectivity pattern is 'hexagonal', so every neuron has at most six neighboring neurons. For each data point in the dataset on which this neural grid is trained a winning neuron is selected, this is the neuron by which the neuron weight vector and data point have minimal distance, in the case of this research the distance metric used is Gaussian [18]. For a winning neuron the weights, the weights within its neighborhood according to the neighborhood function. Neighboring neurons are selected first at its largest neighborhood dimension. For example, all topologically connected neurons within two hops can be updated, that is to say 19 neurons are updated (sum of neighbors within 2 hops: 12, 1 hop:6, 0 hops:1). As time t increases the dimension of topologically neighboring neurons decays to one hop away from the winning neuron. Eventually the neighborhood is reduced to only the winning neuron being updated. The value update is determined by the learning rate which is influenced by the epoch parameter and decays over time. This process iterates until the desired number of epochs is reached and with the correct number of iterations convergence is reached.[18] With this iterative training process for the SOM eventually the network organizes the network to represent the original datas topology and eventually given enough time to anneal the neighborhood size, neurons will respond to increasingly

specific characteristics of the data. This algorithm has seen success in a variety of areas, most significantly in the case of this study's NLP approach to log analysis is the success of SOM in text mining [19].

3.3.5 Hierarchical Self-Organizing Maps

As for a hierarchical SOM (HSOM), it is a multi-layered representation of the previously described SOM implementation. This is an adaptation in which a single layer SOM is analyzed post-training to steadily build more abstract features as the number of SOM layers increase [17] [9] [33]. That is to say, the hypothesis is that features learned at the initial layers of a hierarchy may still be interpreted in terms of recognizable basic measured properties, whereas features at the highest level in the architecture will capture aspects synonymous with normal or attack behaviors. The general method of building a hierarchical SOM is described below:

1. Train First Layer SOM: Train the first layer and label the neurons, while gathering metrics on the distribution of ANOMALY:NORMAL hits each neuron receives.

2. Select Mixed Neurons: Using the metrics gathered while labeling the neuron map, for each neuron identify which ones are mixed neurons and should be opened into the second layer based on a selection criteria composed of a hit count, and ratio of ANOMALY:NORMAL. For the purposes of this research the criterion was to have at minimum 5 hits, and maximum 80% of one class.

3. Split Training Set: Split the training set based on whether it hits a mixed neuron and map it to the neuron index, to create individual training sets for each second layer map.

4. Train Second Layer: Using each individual training set begin training and labeling each of the second layer SOMs.

This approach requires some labeled data in order to make the selection of mixed neurons and as such strays from the unsupervised focus of this research. However, labeled data is only important in training, for testing the list of mixed neurons identified during training is referenced when a validation data point hits a neuron in order

to identify if it should then go through to the second layer SOM trained for that mixed neuron to identify its predicted label.

3.4 Analysis and Visualization

In this research, two visualization algorithms are utilized, SOM Hit maps and t-SNE, to discover the clusters or distributions of the data using different representations of the syntactic and semantic feature sets. The t-SNE algorithm is a non-linear dimensionality reduction technique which attempts to preserve small pairwise distances [37]. Whereas SOM visualization is built-in with the SOM learning algorithm. A feature of the the SOM algorithm employed in this study is the ability to generate the neuron map it produces, which can then be used to visualize high dimensional data on a two-dimensional plane [18]. The SOM maps can also use hit frequency counts for the neurons in order to produce hit maps, which provide information about the topology of the map. The t-SNE and SOM visualizations using hit maps are shown in Figure 4.9.

3.4.1 t-SNE: t-Distributed Stochastic Neighbourhood Embedding

The t-Distributed Stochastic Neighborhood Embedding algorithm is a non-linear dimensionality reduction algorithm typically used for visualization, feature extraction and clustering [36]. The algorithm is particularly useful in visualizing high-dimensional data in low dimensionality representations while preserving the distances between widely separated points. This preserves the underlying structure of the data. The following is a summarization of the different steps in the t-SNE algorithm.

1. Compute Pairwise Similarity: Compute the pairwise similarity measure between every data point in the input dataset. This is typically accomplished by performing using Gaussian kernel on the euclidean distances of each pair of data points

2. Initialize Low-Dimensional Space: Initialize a low-dimensional space with randomized value for each data point.

3. Compute Pairwise Similarity of Low-Dimensional Space: Using the same pairwise similarity algorithm compute the pairwise similarity of the low-dimensional representation.

4. Compute similarity mismatch: Using the pairwise similarity matrix of the low-dimensional and high-dimensional spaces compute the differences using Kullback-Liebler divergence. Kullback-Liebler divergence measures the differences between probability distributions.

5. Optimize Low-Dimensional Space: Adjust the position of the low-dimensional space to minimize KL divergence using gradient descent.

6. Repeat: Perform the last three steps until KL convergence is at a minimum. The goal of this optimization is to find a low-dimensional space which preserves the pairwise similarities of the orginal dataset as much as possible.

t-SNE aims to preserve the pairwise similarities of the original dataset by mapping a low-dimensional space to the high-dimensional representation of the data [36]. This process of iterating to find an optimal low-dimensional space is computationally intensive, particularly for large highly dimensional datasets. The algorithm does produce visualizations can reveal unique underlying structures in the dataset.

3.5 Summary

In this chapter, the overall methodology of the thesis is presented including the datasets, feature extraction techniques, and unsupervised machine learning algorithms used. Also further discussion is provided on the different language models used in this thesis, their respective architectures as well as the fine-tuning approach including the training corpus used. There are four datasets used in this research, namely ISOT, PKDD, AA, and CSIC, to evaluate the various apporaches introduced. It should also be noted here that there is an additional dataset, Harvard dataset, which is only used for the fine-tuning of the chosen language model. Collectively, these datasets represent a unique selection of application logs relating to cloud and web systems. Moreover, the unsupervised learning algorithms used are K-Means, Isolation Forest, Agglomerative Hierarchical Clustering, Self-Organizing Maps, and

Hierarchical Self-Organizing Maps. The feature extraction techniques include syntactic and semantic approaches. The syntactic approach utilizes seven syntactic features extracted from each log line, namely minimum, maximum, and average TF-IDF scores over the terms of the log files. The semantic approach utilizes large language models to extract meaningful sentence embedding representations from the final layer of the neural network. There are five language models employed in this thesis, namely Bert, RoBERTa, DistillToBERTa, GPT2 and GPT-NEO. In short, this research explores how well semantic and syntactic feature extraction techniques work and differ from each other in terms of representing application log files from the lens of security analysis and anomaly detection. Thus, a comprehensive benchmarking approach is employed to evaluate each feature extraction approach using five unsupervised learning algorithms over four publicly available application log files. Moreover, for the semantic approach, five different language models are evaluated with and without fine-tuning in order to understand the strengths and limitations of the semantic feature extraction approach on application logs. Finally, the impact of the log structure is also analyzed from the lens of the semantic feature extraction approach.

Chapter 4

Evaluations and Results

4.1 Metrics

The performance metrics used in this research are F1-score, Precision and Recall. Given all the datasets employed have labels already provided by the datasets authors, the following metrics could be used post training and post testing for measuring the performance for each evaluation done in this thesis. Thus, these metrics could be calculated using the resulting confusion matrixes of the training and test evaluations for each technique used in the thesis.

The F1-score is a traditional method for measuring the model accuracy on a particular dataset, it is computed using Precision and Recall. Precision gives an indication of the percentage of true positives (inclass) which are relevant while Recall gives an indication of the percentage of true positives out of all the true positives available in the data. The formulae for Precision, Recall and F1-score are given in the following equations 4.1, 4.2 and 4.3 [25].

$$P = \frac{TP}{TP + FP} \tag{4.1}$$

$$R = \frac{TP}{TP + FN} \tag{4.2}$$

$$F1 = \frac{2PR}{P + R} \tag{4.3}$$

Macro F1-Score is used to calculate the unweighted mean of all per-class F1-scores, calculated by summing the F1-score for each class label over the total number of classes n. This is calculated with the following formulae in Eq. 4.4.

$$F1_{macro} = \frac{\sum F1_{class}}{n} \tag{4.4}$$

Weighted F1-Score is used to calculate the per class weighted mean of all F1-scores, that is to say the average F1-score where per class proportion, S of the dataset is considered. This is calculated with the following formulae in Eq. 4.5.

$$F1_{weighted} = \sum \frac{F1_{class}}{S_{class}} \tag{4.5}$$

Micro F1-Score is used to calculate the total average F1-score by first calculating the total true positives, false negatives and false positives across all classes and computing the F1-score using the following formulae in Eq. 4.6.

$$F1_{micro} = \sum \frac{2P_{cumulative}R_{cumulative}}{P_{cumulative} + R_{cumulative}} \tag{4.6}$$

In the remaining of the thesis, the decision of "best" performing algorithm is based on the Weighted F1-scores obtained during evaluations.

4.2 Evaluation Structure

4.2.1 Syntactic vs Semantic Approach Evaluation

For the purposes of exploring the anomaly detection with either the syntactic approach or the semantic approach to log data feature extraction, a benchmark using different unsupervised machine learning techniques was performed. This study provided a comparison of the semantic and syntactic feature representation of the four different datasets, ISOT, PKDD, CSIC and AA, after using five different unsupervised machine learning algorithms for anomaly detection. The unsupervised machine learning techniques were trained and fit using a 75% training set split of the original datasets, and then subsequently tested using the remaining portion, 25%, of each dataset.

The classification results for the models was calculated in the case of the K-Means, and Agglomerative algorithms by labeling the clusters using majority vote, the clusters identified by the model were labeled exclusively based off of the class with the prevailing amount of representation in it during training.

In the case of Isolation forest a classification report was generated by identifying if ANOMALY labeled data was classified as abnormal, and much the same for NORMAL data.

For SOM algorithms classification metrics were calculated by labeling each neuron based on the labels of the data which hit the neuron during training, a majority vote algorithms is used here as well to identify the winning label for a given neuron. Much like a single layer SOM the HSOM used a majority vote for labeling neurons in the second layer. In both SOM implementations an attempt was made to mitigate the number of empty labeled neurons, once the map had been labeled using majority vote, an algorithm would label empty neurons based on the nearest neighbors label.

In order to effectively compare the different feature extraction techniques the time required to extract the information and build each model was also collected. Feature extraction time and inference time are important factors in log analysis since the veracity, velocity and volume of the data is quite high, so being able to rapidly identify abnormal events in logs is important.

4.2.2 Large Language Model Log (LLM) Evaluation

Exploration of the semantic feature extraction technique in the first portion of this study and the wide variety of language models that are publicly available led to an interest in ascertaining which language models were more appropriate for the unique task of log analysis and anomaly detection. Thus another study was performed in which five large language models were evaluated in their ability to extract meaningful data in the sentence embedding representations of log lines pulled from three of the datasets, AA, PKDD, CSIC. In this study only the K-Means unsupervised machine learning algorithms was used to perform the anomaly detection component. The experiment structure was as follows:

1. Extract sentence embeddings from different language models, BERT, RoBERTa, DistilRoBERTa, GPT-2, GPT-NEO. Record the time taken to extract features.

2. Split the sentence embedding representation into 75% training and 25% test.

3. Train K-Means clustering algorithm using the training set split. Record the time taken to generate the K-Means model.

4. Label K-Means clusters using majority vote.

5. Evaluate model performance with testing set.

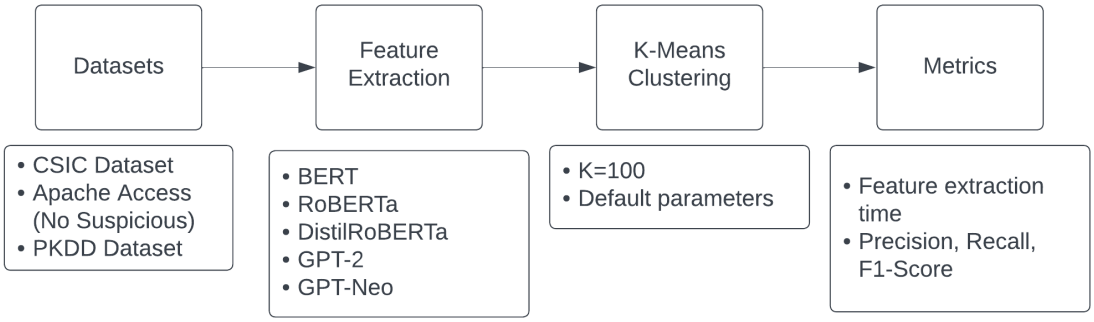

Figure 4.1: LLM Baseline Experiment Methodology

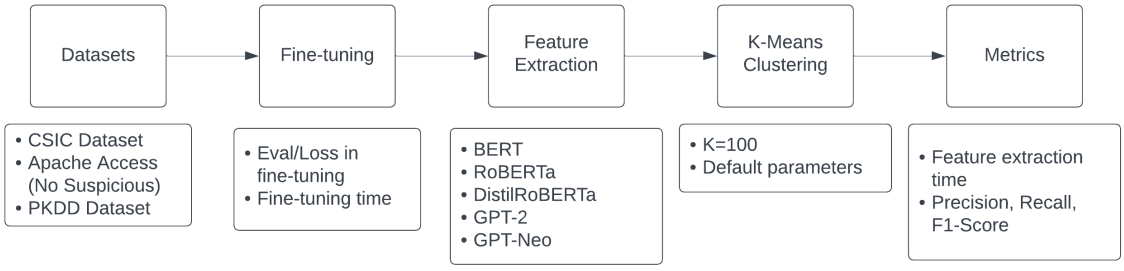

Figure 4.2: LLM Fine-Tune Experiment Methodology

Performing these steps on each dataset across the five different models produced a baseline for which models without any further adaptation can perform the task of log analysis. Though this leads into the next important question which is how can the language models be adapted to better perform log analysis.

4.2.3 LLM Baseline vs Fine-tuned Evaluation

Establishing a baseline for these language models provides an opportunity to explore the benefits of fine–tuning of language models specifically for log analysis. Fine-tuning of language models is often referred to as domain adaptation and describes the method of adapting the large language models using different model dependant techniques, most commonly masked language modelling. In masked language modeling the language model is fed a masked corpus of text where the model is tuned to make better predictions of the masked keywords. This process can often provide a 1-3% boost in domain specific tasks [8]. For the purposes of this study the harvard Apache access dataset [39] had 100,000 random logs sampled from it and used as the fine-tuning dataset for all five of the different language models outlined previously. The same experiment setup as in the previous section was then repeated using the now fine-tuned language models in order to identify which models provided the largest fine-tuning benefits. The same data points were collected so as to provide a comparison between baseline language models and their fine-tuned counterparts.

4.2.4 Evaluation on the Effect of the Log Structure

The structures of the different log datasets vary significantly, AA has a more typical Apache web server log line structure while PKDD and CSIC have 53 and 15 features respectively, which could be taken from network and application logs, and follow a key value pairing structure for each log line. Thus PKDD and CSIC produce more verbose and mixed single line log lines when fed into the language models, cursory exploration of the baseline and fine-tuned language models identified AA as the best performing dataset and so a new question was raised. How does the structure of the log line impact the effectiveness of semantic feature extraction? In order to explore this question the PKDD and CSIC dataset were converted into Apache like log lines following the typical structure of default Apache webserver logs. These two

Table 4.1: HyperParameter Setting

Algorithm	PKDD	ISOT	AA	CSIC
HSOM	Top Grid = 40*40, Bottom Grid = 6*6, Epochs = 10000			
SOM	Grid = 40*40, Epochs = 10000			
K-Means	k=100	k=100	k=100	k=100
Agglomerative	c=10	c=50	c=100	c=50
Isolation Forest	i=0.3	i=0.3	i=0.15	i=0.3

Apache like representations of PKDD and CSIC would be known as apachePKDD and apacheCSIC. They were both evaluated using the language model benchmarking experiment methodology with both fine-tuned and baseline language models.

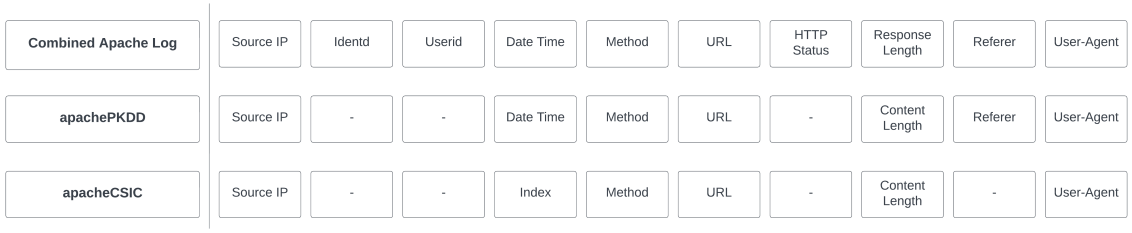

Figure 4.3: Construction of combined Apache access log for PKDD and CSIC datasets

The performance of each of these models is evaluated by labelling the clusters post-training. This results in an unsupervised model which can be used for prediction on the test datasets. As mentioned before, the training/testing split of 75% to 25% is used. Since all three datasets are unbalanced datasets, weighted precision, recall and F1-score are used for measuring the performance. The hyperparameters are tuned on each model for each dataset. Table 4.1 shows the hyperparameters that are chosen.

4.3 HyperParameters for Learning Algorithms Used

The following section outlines the process of how the hyperparameters are tuned on each model for each dataset. Table 4.1 shows the hyperparameters that are chosen.

4.3.1 K-Means Clustering

The best value of k in k-means was derived through generating k-means models with increasing k number of clusters and comparing training classification performance

across the resulting models. An elbow study was performed for this model in particular in order to derive an optimal value of k across all datasets. The figures 4.4, 4.5, 4.7, and 4.6 show the results of this study. The power law curve was used to approximate the change in F1-score as k increases consistently showing a plateau in difference between 90 and 110 clusters acrosss the different datasets. Thus, k value 100 was selected for all experiments in which K-means was employed. The Table 4.1 reflects this choice.

4.3.2 Agglomerative Clustering

The best value of c in the agglomerative hierarchical clustering algorithm was derived through evaluating the performance of agglomerative models with increasing values of c. Weighted F1-score was used here again to identify the optimal value of c for each feature set across all the datasets. Table 4.1 identifies these best performing values of c.

4.3.3 Isolation Forest

The primary tunable parameter in this ensemble technique is the contamination ratio which identifies the proportion of anomalies present in the dataset. In order to derive this parameter the proportion of ANOMALY labeled data points is calculated. This metric is derived for the feature set on model generation and the resulting contamination settings can be seen in Table 4.1.

4.3.4 Self-Organizing Maps

SOMs have a variety of different parameters for tuning which can impact its overall performance. Exploration of varying map/grid sizes with both the syntactic and semantic feature sets across all the datasets identified the parameterization outlined in Table 4.1 as consistently high performing.

4.3.5 Hierarchical Self-Organizing Maps

The hierarchical SOMs expand on the single layer models parameterization previously outlined with new second layer related parameters. These include second layer

map size, topology, neighborhood function and most importantly the mixed neuron selection heuristic which is used to identify mixed neurons which are then candidates for further exploration in the second layer. This heuristic evaluates the ratio of ANOMALY:NORMAL data hitting that top level neuron and the total hit count of that neuron. Hierarchical SOM parameterization is seen in Table 4.1.

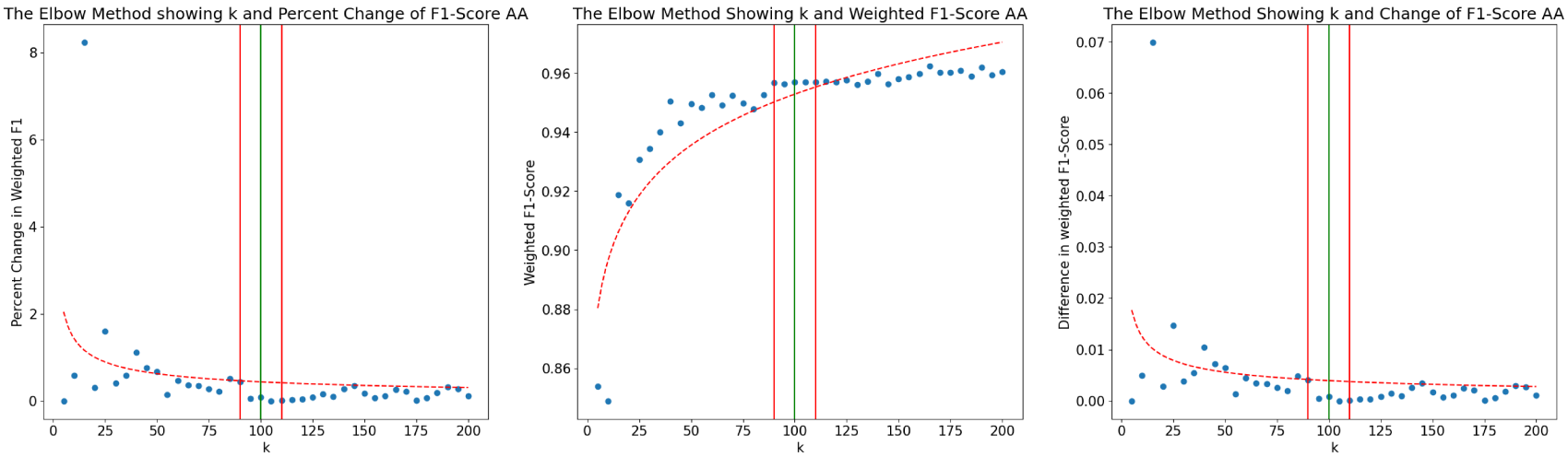

Figure 4.4: Elbow study of K-Means Apache Access

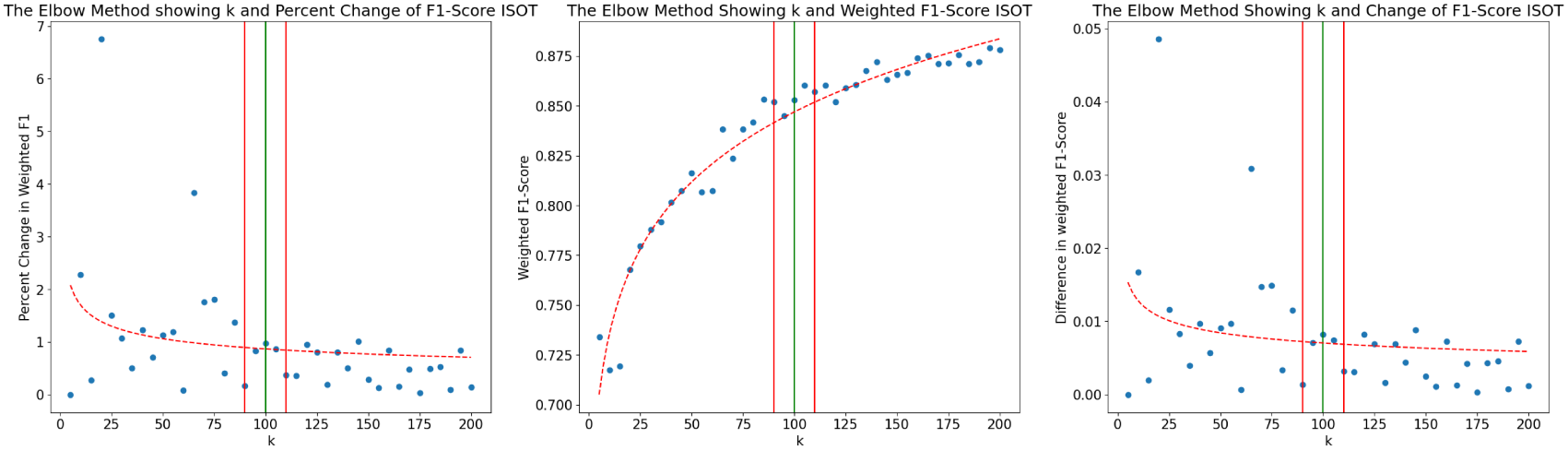

Figure 4.5: Elbow study of K-Means ISOT

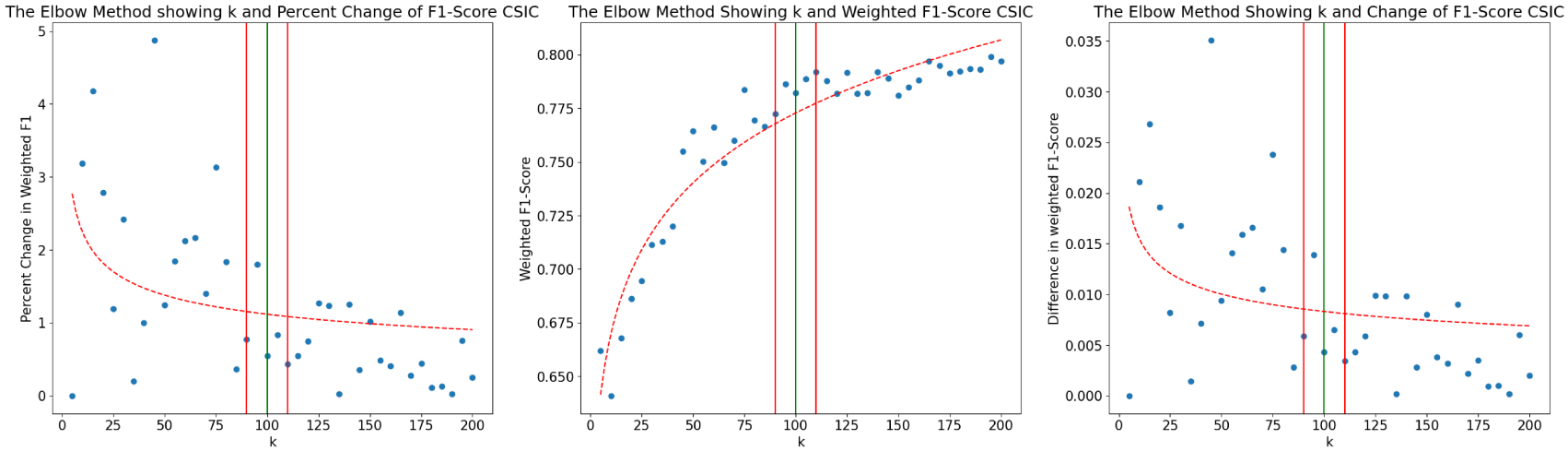

Figure 4.6: Elbow study of K-Means CSIC

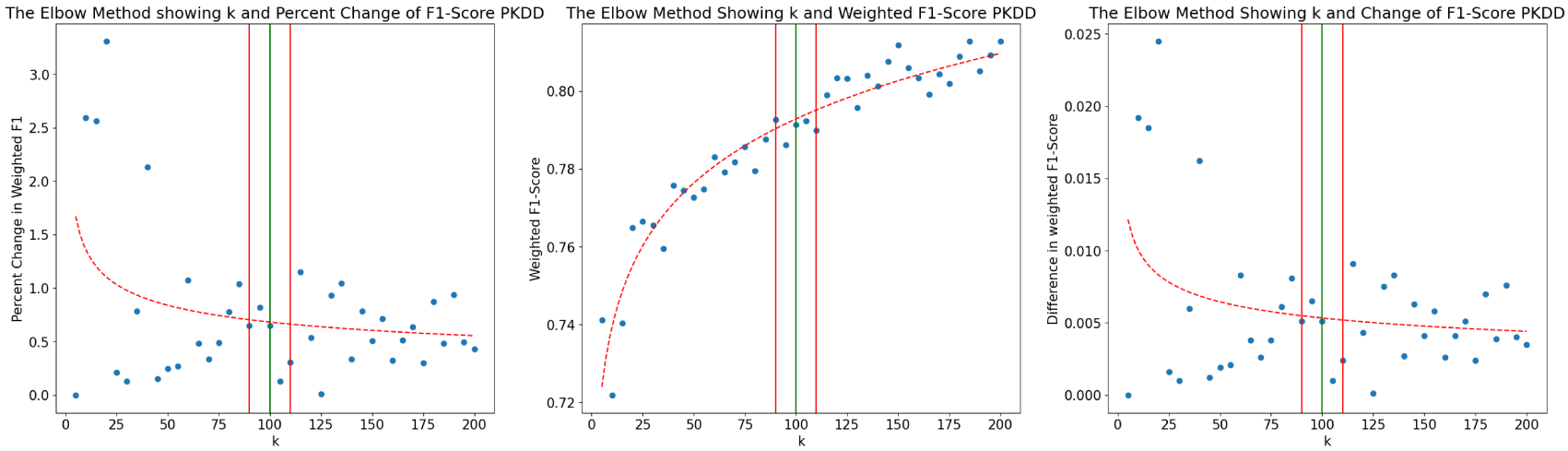

Figure 4.7: Elbow study of K-Means PKDD

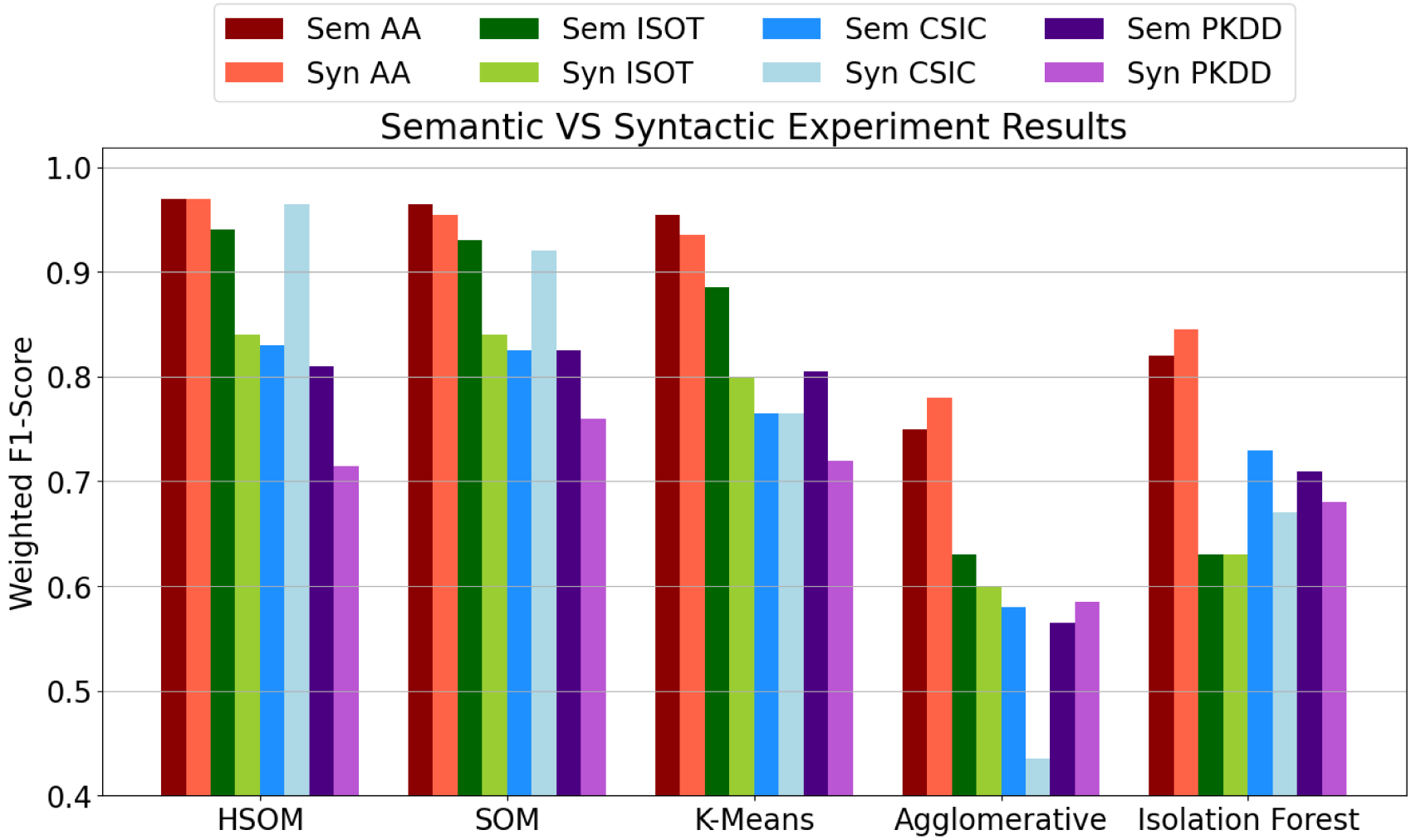

Figure 4.8: Semantic (Sem) vs Syntactic (Syn) experiment results: Weighted F1-Score

Table 4.2: Model Training Time: Semantic VS Syntactic approaches (Seconds)

	Model Training Time							
	AA		**ISOT**		**CSIC**		**PKDD**	
Unsupervised ML	Syntactic	Semantic	Syntactic	Semantic	Syntactic	Semantic	Syntactic	Semantic
Isolation Forest	19	58	37	80	44	73	30	73
Agglomerative	25	197	53	420	60	471	54	429
K-Means	3	35	4	60	5	84	5	102
SOM	346	1,300	458	1,847	480	2,068	531	1,906
HSOM	431	1,401	631	2,183	535	2,739	673	2,524

Table 4.3: Semantic VS Syntactic Model Performance Results: Weighted F1-Score

	Testing Weighted F1 - Score							
	AA		**ISOT**		**CSIC**		**PKDD**	
Unsupervised ML	Syntactic	Semantic	Syntactic	Semantic	Syntactic	Semantic	Syntactic	Semantic
Isolation Forest	0.845	0.82	0.63	0.63	0.67	0.730	0.680	0.710
Agglomerative	0.75	0.775	0.600	0.630	0.435	0.58	0.585	0.565
K-Means	0.935	0.955	0.80	0.885	0.765	0.765	0.720	0.805
SOM	0.955	0.965	0.840	0.930	0.920	0.825	0.760	0.825
HSOM	0.970	0.970	0.840	0.940	0.965	0.830	0.715	0.810

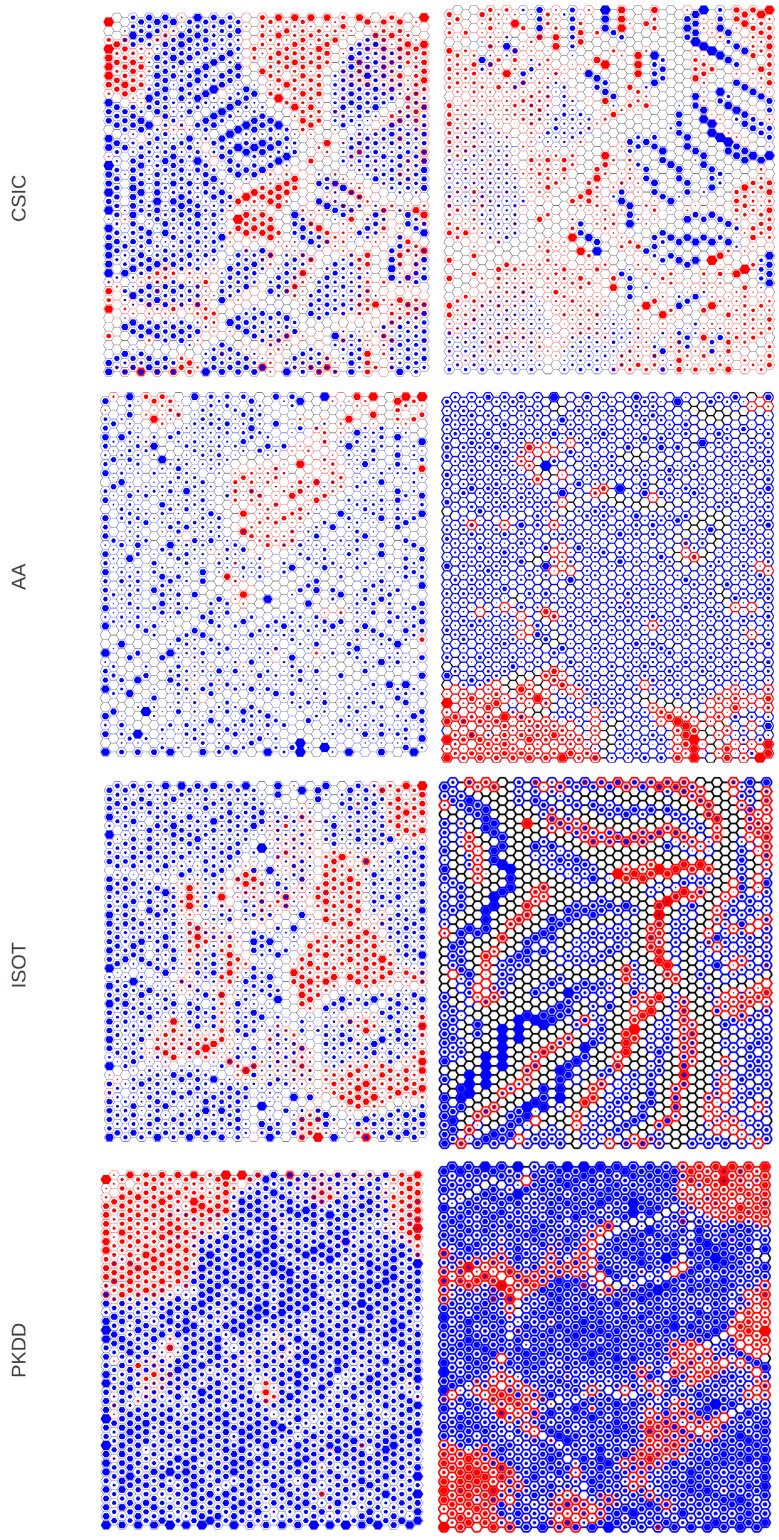

Figure 4.9: SOM Hit Map Visualizations of the datasets

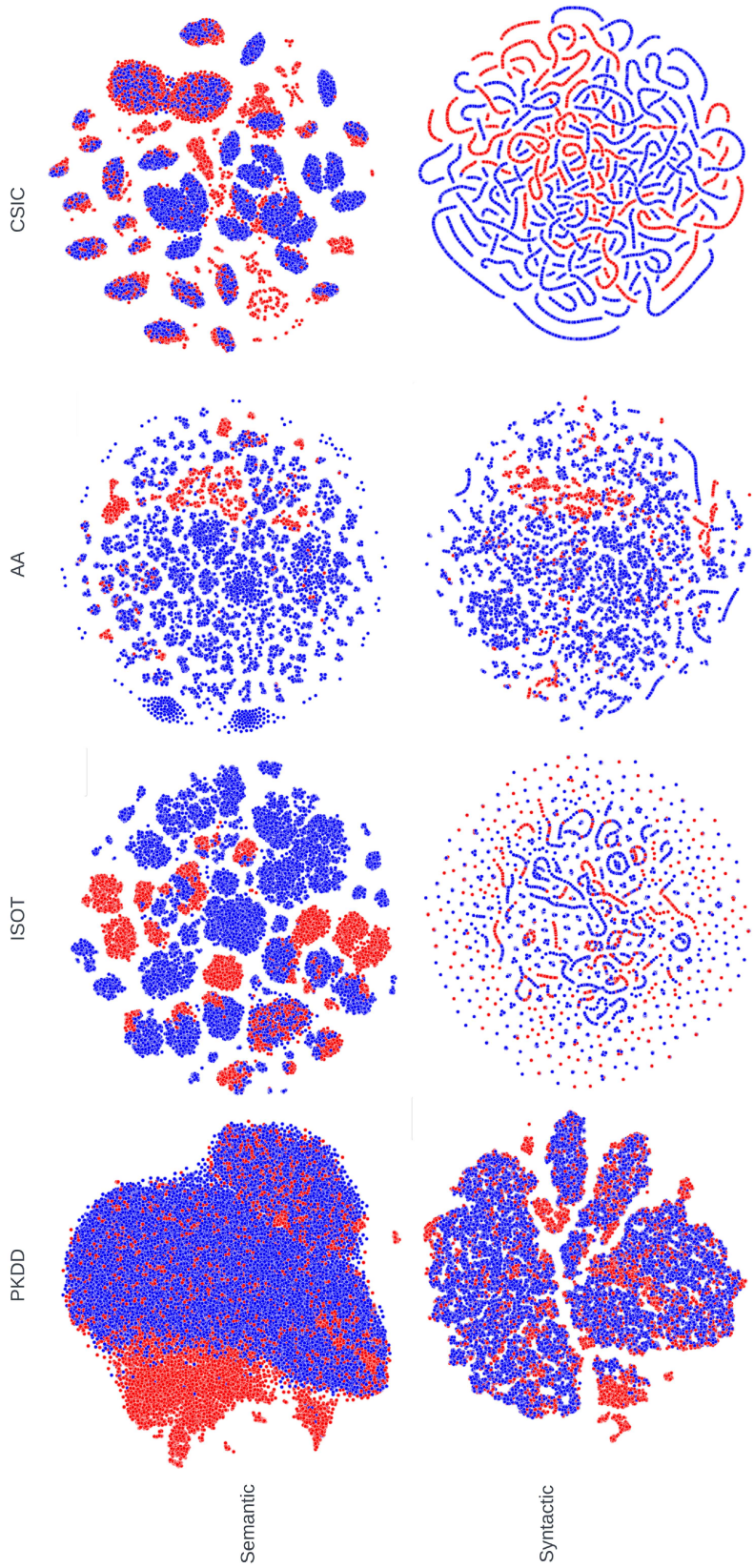

Figure 4.10: t-SNE Visualizations of the datasets

Table 4.4: Semantic vs Syntactic Feature Extraction Times (Seconds)

	Feature Extraction Time (Seconds)			
Extraction Technique	**AA**	**ISOT**	**CSIC**	**PKDD**
Semantic	44	60	138	217
Syntactic	5	4	11	27

4.4 Semantic vs Syntactic Experiments

The following section outlines the results and discussion/analysis of the semantic vs syntactic experiments.

4.4.1 Results

Figure 4.8 shows the weighted F1-score classification results for the unsupervised machine learning techniques employed in this work for both semantic and syntactic feature sets of all four datasets. The results indicate that in AA, ISOT, and PKDD a semantic feature extraction technique is more successful in disambiguating anomalous events. All three datasets show that semantic feature extraction outperforms relative to syntactic independently from the unsupervised machine learning model used. In contrast, for the case of the CSIC dataset the syntactic method outperforms it's semantic counterpart when using the SOM and HSOM algorithms. The difference of 13% between the CSIC syntactic and semantic when using HSOM and 10% in SOM is notable since it represents the largest difference in performances between the two feature extraction techniques across all results excluding the abnormally low syntactic performance of agglomerative clustering for the very same dataset. Another observation is the unique case of the PKDD dataset, where semantic feature extraction sees a lead of 10% in HSOM, 6% in SOM, and 8% in K-Means over syntactic. The ISOT results also reflect a similar favor for semantic feature extraction with 10%, 9%, and 8% in HSOM, SOM, and K-Means respectively.

The performance of isolation forest and agglomerative clustering is low across the board with unremarkable results relative to the K-Means model which according to the literature, in particular Copstein et al. represents the current best performing unsupervised algorithm for short syntactic feature extraction. Since the syntactic

feature set does not outperform K-Means on these models and presents no additional benefits, such as visualization through the SOM hitmaps, they can be comfortably overlooked as strong candidates for effective anomaly detection using these feature extraction techniques. The semantic results for these models also pale in comparison to the K-Means, HSOM, and SOM results.

Upon inspection of the t-SNE and SOM hit map visualizations for all of the datasets seen in Figures 4.10 and 4.9, in both the semantic and syntactic forms, a similar story unfolds. The semantic feature sets in t-SNE form see much more consistent separation between anomalous and normal labeled data. This is particularly the case for the ISOT dataset, where the semantic feature set represents presents with clearly defined clusters of anomalous data in the t-SNE, and good separation in the SOM hit map. Empty neurons in the SOM hit map for the semantic ISOT featureset contribute to this separation, and outline defined islands of neurons with high anomalous activity. The syntactic ISOT SOM and t-SNE visualizations by contrast see significant amounts of overlap and ambiguity. The string-like formations, and tiny mixed clusters in the t-SNE plot point to low variance in the short syntactic representation of the dataset. This is also seen in the SOM hit map representation where a lot of highly mixed activity is occurring on the labeled neurons.

Inspection of the CSIC syntactic featuresets visualizations provides some insight on the distribution of normal labeled data since both the t-SNE and SOM hit map show significant overlap of normal data. The SOM hit maps in particular show that while the dataset has a 70/30 normal/anomaly distribution the number of normal labeled neurons does not match, indicating the syntactic representation of CSIC has a low degree of variance. The t-SNE plot presents an interesting contrast to that of the ISOT one, in that it shows the same string-like formations, unlike ISOT these formations have formed as clusters of similarly labeled data types. The disambiguation of anomalous behavior seen in the visualizations of CSIC in its syntactic representation reflect the results of the learning models, particularly in the case of the SOM and HSOM.

The PKDD visualizations for both syntactic and semantic feature sets make it immediately apparent that this dataset is noisy, and there is a challenge in effectively

separating normal and abnormal behavior. The t-SNE plot of the syntactic representation of PKDD shows a haphazard mix of normal and anomalous data for the majority of the plots, with few individual clusters of anomalous data. The semantic model reflects this mixed bag of data, but with a unique cluster of entirely anomalous data floating left. The SOM hit maps for syntactic and semantic also reflect this very mixed activity. However, similarly to the t-SNE plot the semantic feature set does have some grouping of anomalously labeled neurons in the top left. The mixed behavior reflects the learning model results and is likely due to the loss of information in anonymization.

The Apache Access visualizations compliment the learning model performance results in that clusters can be clearly identified. The t-SNE visualizations for both syntactic and semantic feature sets of AA have distinguishable clusters. Additionally, the SOM hit maps indicate disambiguation of anomalous behavior with obvious isolated groups of neurons with strong anomalous activity. Not unsurprising given the AA dataset is the consistently best performing dataset in this research with both syntactic and semantic learning models performing above 0.90 F1-score.

4.4.2 Case Study: ISOT Log file

In the context of the ISOT dataset it becomes immediately clear through the performance results given in Figure 4.8 and dataset characteristics in Table 3.6 that log type influences the success of these NLP inspired techniques. The ISOT dataset being the output of TCPdump executed on cloud infrastructure provides highly structured log data. Some examples of ISOT logs can be seen in Figure 4.11. Thus this structure does not provide unique Syntactic features that are differentiable. The information columns in the log line responsible for limited separation in the syntactic approach are highlighted in green and yellow in Figure 4.11, green indicating that that data is either completely ubiquitous or has very little variation and yellow indicating data which contributes very little because they are trivially unique to that log line, from a syntactic standpoint. The Semantic approach on the other hand is able to extract more meaningful information from these records. In this case, the language model DistilRoBERTa provides a feature set of 768 embeddings. The difference between the semantic and syntactic features in terms of its variation is shown visualizations

in Figure 4.9 and 4.10. The logs presents information on the different ports and IPs as well as packet size and flag information. These log representations provide limited meaningful syntactic information because the information is either highly specific and therefore entirely unique to that log (ie. packetSize, sequenceNumber, ackNumber) or almost entirely ubiquitous throughout the whole dataset (ie. flags, fragmented, protocol). Therefore of the 10 data points per log line, three of them will likely occupy the Minimum-TF-IDF feature, three of them will likely occupy the Maximum-TF-IDF feature, and the remaining four data points may influence the average TF-IDF feature but not as significantly as the former six. The remaining general syntactic features of the log line (AlphaNumericRatio, AverageCharacter, CharacterCount and WordCount) also provide minimal unique log line information because the number of words, characters and alphanumeric ratio are fairly consistent due to the simple rigid structure of TCPdump information. This essentially means that the process of identifying anomalous events in this form of log data for the proposed syntactic approach hinges on the two features; average TF-IDF and average character, even in these features the variation is highly limited, again due to the simple log line representation. This limitation of the syntactic approach is clearly visualized in its T-SNE visualization and SOM hit map in figures 4.10 and 4.9. The extracted feature data sees a significant amount of overlap thus creating the almost 'string-like' representation in those visualizations. The semantic approach on the other hand is able to evaluate the entire context of the log line enabling it to more effectively derive information from the different components. DistilRoBERTa, as a BERT-style model provides a much more complex featureset (768 embeddings) which is then able to take full advantage of the entire log line, thus enabling the semantic approach to outperform syntactic features. The difference between the semantic and syntactic features in terms of its variation is clearly seen in the contrast between the T-SNE and SOM visualizations in figures 4.10 and 4.9.

4.4.3 Discussion

In summary thus far this study has explored the utilization of four different unsupervised machine learning techniques, using two NLP inspired feature extraction techniques on four distinctly different datasets, where each represents a different type of

protocol	sourceIp	sourcePort	destIp	destPort	size	fragmented	seqNumber	ackNumber	flags
tcp	172.16.1.28	22	172.16.1.24	33642	1258	FALSE	1517383816	1419430660	ACK PSH
tcp	172.16.1.24	22	134.87.154.134	59526	1608	FALSE	736934705	-1737241340	ACK PSH
tcp	172.16.1.24	22	134.87.154.134	59502	344	FALSE	-1944819960	138274026	ACK PSH
tcp	172.16.1.28	22	172.16.1.24	32900	1450	FALSE	951876763	1430854278	ACK
tcp	172.16.1.24	37434	142.104.64.196	22	382	FALSE	2035867867	1223010965	ACK PSH
tcp	172.16.1.24	33642	172.16.1.28	22	52	FALSE	1419434656	1523151150	ACK
tcp	172.16.1.24	22	134.87.154.134	59533	2628	FALSE	676078423	-198765918	ACK
tcp	172.16.1.28	22	172.16.1.24	33642	1274	FALSE	1216667112	1419221788	ACK PSH
tcp	172.16.1.24	22	134.87.154.134	59526	2680	FALSE	733497145	-1737243752	ACK PSH
tcp	172.16.1.19	22	134.87.154.134	59513	2488	FALSE	-713646544	304653219	ACK PSH
tcp	172.16.1.28	22	172.16.1.24	33642	1306	FALSE	1307377752	1419284824	ACK PSH

= Ubiquitous = Erratic

Figure 4.11: Example log records from ISOT dataset

security log data, namely regular application logs, anonymized and non-anonymized network/application logs, and a cloud network traffic log. The results indicate that AA (regular application log) shows that both feature extraction techniques provide high F1-scores independent of the unsupervised machine learning algorithms used, though the semantic feature extraction technique outperforms marginally. The HSOM, SOM, and K-Means algorithms perform the best with AA, and also represent the three best performing models for all four datasets. Moreover, the results on PKDD continue this trend where the semantic feature extraction algorithm outperforms the syntactic with high F1-scores in HSOM, SOM and k-means, even with fairly mixed hit map and t-SNE visualizations. The low syntactic results and muddied visualizations for both featureset are likely due to the anonymization of the dataset, which makes TF-IDF analysis difficult to perform and disambiguation of anomalous behavior arduous. The results for the ISOT dataset indicate a clear advantage of the semantic feature set on highly rigid structured logs over the syntactic methodology. Semantic outperforms syntactic independently from the unsupervised machine learning algorithm with the ISOT dataset. The benefits of log structure in semantic feature extraction was identified in this dataset. Finally, the CSIC dataset sees remarkable learning model performance with syntactic feature extraction, with HSOM and SOM results with a noticeable 13-10% gap. This dataset shows how the syntactic feature extraction technique can successfully disambiguate abnormal behavior by identifying the abnormal components of each log line. Another consideration worthy of discussion is the feature extraction and model training times that come with these

different algorithms. The feature extraction times can be seen in Table 4.4, and model training times can be seen in Table 4.2. The semantic feature extraction technique requires significantly more time to gather the sentence embedding representation from the large language model than it takes to compute syntactic characteristics. The sentence embedding representation is also many times larger at 768 dimensions than that of the short syntactic representations with only 7 dimensions. This larger feature set size means model creation time is exponentially longer, as seen in Table 4.2. The semantic feature set may outperform the syntactic feature sets in three out of the four dataset but it's sometimes marginally higher performance comes at a much higher cost. Alternately, the syntactic feature extraction techniques relies on having already collected a dataset in order to compute the TF-IDF values for tokens, whereas the semantic feature extraction using a pre-trained large language model can perform a cold start with no data required to begin successfully extracting the sentence embedding representation of the log line. Evaluation of both the computational costs, and performances indicate that K-Means remains a strong performer with a very low cost, not outperforming the SOM and HSOM methods in terms of performance metrics but certainly cost to performance ratio. The SOM and HSOM algorithms on the other hand provide an innate advantage of also producing much need visualization and interpretability to the log data in the form of SOM hit maps.

The selection of a better performer between the two feature extraction techniques is another debate entirely. The semantic feature extraction technique outperforms the syntactic in three out of four of the datasets but does so at a relatively high computational cost, with sometimes unconvincing marginal improvement on syntactic performance, particularly in the case of the AA dataset. Since semantic feature extraction requires usage of large language models to extract embedding representations an opportunity presents itself for continued exploration of how to best adapt these models for the use of log analysis and anomaly detection. Domain adaptation of the underlying language model may yield more significant improvement over that of the syntactic extraction technique. Additionally the unique observation about the structure of datasets being important to the success of syntactic and semantic feature extraction techniques brings into scope some new avenues for exploration. The semantic feature extraction technique and by extensions the language model which

Table 4.5: LLM Feature Extraction Times

	Feature Extraction Time (Seconds)		
Language Model	**AA**	**CSIC**	**PKDD**
Bert-base-cased	83	266	387
RoBERTa	73	239	385
DistilRoBERTa-base	44	138	217
GPT-2	79	263	657
GPT-NEO	727	2,581	5,960
apacheBERT	82	267	384
apacheRoBERTa	91	237	382
apacheDistilRoBERta	45	138	213
apacheGPT-2	80	265	656
apacheGPT-NEO	807	2,643	6,018

underlies it clearly benefits from a more consistent input sequence structure in the case of the ISOT dataset, would this also be the case for the hybrid network/application log sets of PKDD and CSIC? Further analysis follows in the sections below.

4.5 LLM Experiments

In order to explore language model domain adaptation for the purposes of log analysis, this study has evaluated five different transformer architecture language models on three of the datasets, specifically the three application log datasets, AA, CSIC, and PKDD. The five language models are BERT, RoBERTa, DistilRoBERTa, GPT-2, and GPT-NEO. The literature for benchmarking multiple language models in semantic analysis of security logs for the purposes of anomaly detection is underdeveloped, thus initially it was important to establish a baseline for each model. This meant extracting the model-specific sentence embedding representation of each dataset, and then training a k-means model for anomaly detection on each of these representations. The results of this set of experiments can be seen in Figure 4.7, and the computational cost for extracting the sentence embedding can be seen in Table 4.5.

4.5.1 Results

All five language models produce a unique sentence embedding representation of the logsets, and therefore produce differing results in the performance of the learning algorithm. For the case of the AA dataset, which was consistently the best performing

Table 4.6: LLM Fine-Tuning times

LLM	**Fine-tuning Computation Cost (Seconds)**
Bert-base-cased	8,739
RoBERTa	7,099
DistilRoBERTa-base	7,073
GPT-2	6,944
GPT-NEO	41,531

dataset in the previous set of experiments, BERT, RoBERTa, and DistilRoBERTa which are all BERT-style models see good results, all above 0.95 in weighted F1-score, and relatively low computational cost. The DistilRoBERTa model sees the fastest feature extraction time with 44 seconds, likely owing to its smaller model size being a distilled language model. GPT-2, and GPT-NEO perform the worst in F1-score in this dataset, with below 0.95, and the GPT-NEO feature extraction time peaks at 727 seconds, see Table 4.5. The slow feature extraction time of the GPT-NEO model is due to its much larger model size, with 1.3 Billion parameters, see Table 3.7. This GPT-NEO model also produces a larger sentence embedding vector with 2048 features as opposed to the rest of the models which generate 768 features. This larger vector space leads to a more expensive model creation time, see Table 4.5.

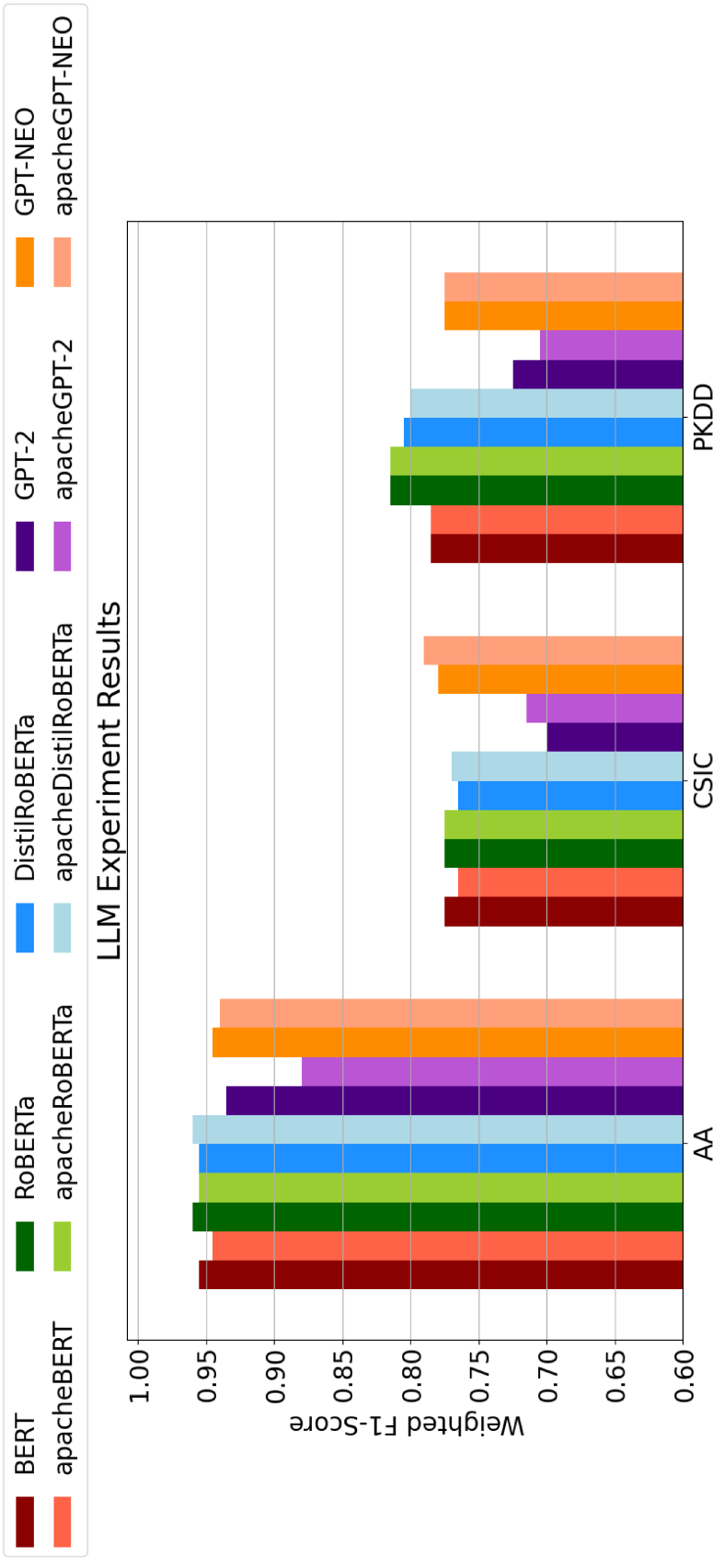

Figure 4.12: Baseline Fine-Tuned LLM Experiment Results: Weighted F1-Score

Table 4.7: LLM Experiment Results: Weighted F1-Score

	Testing Weighted F1-Score		
Language Model	**AA**	**CSIC**	**PKDD**
Bert-base-cased	0.955	0.775	0.785
RoBERTa	0.960	0.775	0.8150
DistilRoBERTa-base	0.955	0.7650	0.805
GPT-2	0.935	0.700	0.725
GPT-NEO	0.945	0.780	0.775
apacheBERT	0.945	0.765	0.785
apacheRoBERTa	0.955	0.775	0.815
apacheDistilRoBERta	0.960	0.770	0.800
apacheGPT-2	0.880	0.715	0.705
apacheGPT-NEO	0.940	0.790	0.775

Table 4.8: Language model performance on AA dataset

Model	Weighted F1	Fine-Tuned Weighted F1	% Change
Bert-base-cased	0.955	0.945	-1.0%
RoBERTa	0.960	0.955	-0.5%
DistilRoBERTa-base	0.955	0.960	+0.5%
GPT-2	0.935	0.880	-5.5%
GPT-NEO	0.945	0.940	-0.5%

Table 4.9: Language model performance on CSIC dataset

Model	Weighted F1	Fine-Tuned Weighted F1	% Change
Bert-base-cased	0.775	0.765	-1.0%
RoBERTa	0.775	0.775	0.0%
DistilRoBERTa-base	0.765	0.770	+0.5%
GPT-2	0.700	0.715	+1.5%
GPT-NEO	0.780	0.790	+1.0%

Table 4.10: Language model performance on PKDD dataset

Model	Weighted F1	Fine-Tuned Weighted F1	% Change
Bert-base-cased	0.785	0.785	0.0%
RoBERTa	0.815	0.815	0.0%
DistilRoBERTa-base	0.805	0.800	-0.5%
GPT-2	0.725	0.705	-2.0%
GPT-NEO	0.775	0.775	0.0%

Alternatively, the CSIC dataset also seen in Figure 4.12 sees the BERT-style models once again perform well in disambiguating anomalous behavior with the learning model results all 0.77. DistilRoBERTa has the fastest feature extraction time again here with 138 seconds, almost half of the computation cost of the other two BERT-style models, and GPT-2. GPT-2 is the worst performer in this dataset with 0.7 weighted F1-score. GPT-NEO on the other hand has the best performance on this dataset, marginally beating out BERT, and RoBERTa, at the cost of a significantly higher feature extraction time of 2,581 seconds, almost 20 times the computation cost of the DistilRoBERTa model. Model size is clearly a huge factor in sentence embedding extraction.

The PKDD dataset sees the RoBERta, and DistilRoBERTa model outperform BERT, and GPT-NEO, with F1-scores that push over 0.8. DistilRoBERta still has the fastest feature extraction time with 217 seconds. GPT-2 once again underperforms with a 0.725 F1-score. GPT-NEO performs worse than the BERT-style models with 0.775 F1-score, and the feature extraction time is an astounding 5,690 seconds, DistilRoBERTa performs better and accomplishes feature extraction in $\tilde{4}$% of the time.

After establishing the baseline for these models for each dataset, they are each fine-tuned using 100,000 sample log lines from the Harvard Dataverse Apache logset. The fine-tuning time can be seen in Table 4.6. The GPT-NEO model takes 41,531 seconds to perform fine-tuning with only 5 epochs. The same set of experiments is then repeated using these newly fine-tuned models, in order to determine which models are able to adapt best to the task of performing log analysis on application logs.

The performance on the AA dataset with fine-tuned models sees only the DistilRoBERTa model outperform it's baseline performance, and only by 0.5%. More interestingly the fine-tuned GPT-2 model gets $\tilde{5}$.5% worse in performance. Other models like BERT, RoBERTa, and GPT-NEO suffer a performance drop as well but only 1-0.5%. The initial observation here is that fine-tuning does not always yield better task-specific performance, particularly in the case of log analysis.

For the CSIC dataset, the fine-tuned models outperform their baseline models in all but two models, BERT and RoBERTa see a 1-0.5% drop. Remarkably GPT-2 sees

the largest jump in performance with a 1.5% increase. DistilRoBERTa still maintains a 0.5% improvement in performance.

Finally, for the PKDD dataset, none of the fine-tuned models outperform their baseline model performance significantly, marginal improvements or losses are gained with less than 0.5% drop or gains in performance.

4.5.2 Discussion

The baseline results indicate that the BERT-style encoder transformer models outperform the autoregressive decoder transformer GPT models in semantic analysis of these application security logs. While GPT-NEO does perform on par or slightly better in some instances, its feature extraction time is significantly longer. The DistilRoBERTa model performs at near and around the top of the five language models on all three datasets, has half the feature extraction time, and it's fine-tuned model is better for both AA, and CSIC. The smaller model architecture lends itself well to more efficient feature extraction. GPT-2 by contrast underperforms consistently, and the fine-tuning of that model causes more unpredictability, as it drops $\tilde{5}\%$ in performance for AA and improves by 1.5% in CSIC for its fine-tuned variation. The poor performance of GPT-2, and underwhelming performance of GPT-NEO may be in part due to the unidirectional component of the model architecture. BERT-style models seem to lend themselves well to the task of semantic analysis of logs for anomaly detection.

The language model experiments yield more questions than answers, domain adaptation through fine-tuning with another publicly available application logset does not seem to consistently provide an improvement in these models for these datasets. The dataset used for fine-tuning is an Apache-style log, and maintains a consistent structure to that of a default configuration Apache access log, perhaps if the CSIC, and PKDD datasets were structured more similarly to that of the Apache log they would see a better performance. This reflects the second question asked at the end of the syntactic vs semantic experiments. This moves into the next area of study in this research. Explore how re-structuring the datasets improve the language models ability to extract meaningful sentence embeddings and subsequently improve the performance of the anomaly detection.

4.6 Log Structure Experiments

After re-structuring the CSIC, and PKDD datasets to reflect a more Apache style log structure the experiment methodology performed for the LLM benchmarking was performed again. The goal of which was to evaluate the impact of log structure on the semantic feature extraction technique. apacheCSIC, and apachePKDD became the pre-processed Apache-style versions of CSIC, and PKDD, and were as close to the Apache structure of logs as could be reasonably accomplished with the features provided by those datasets. The Figure A.1, and Table 4.11 shows the impact of re-structuring of the datasets on language model semantic feature extraction.

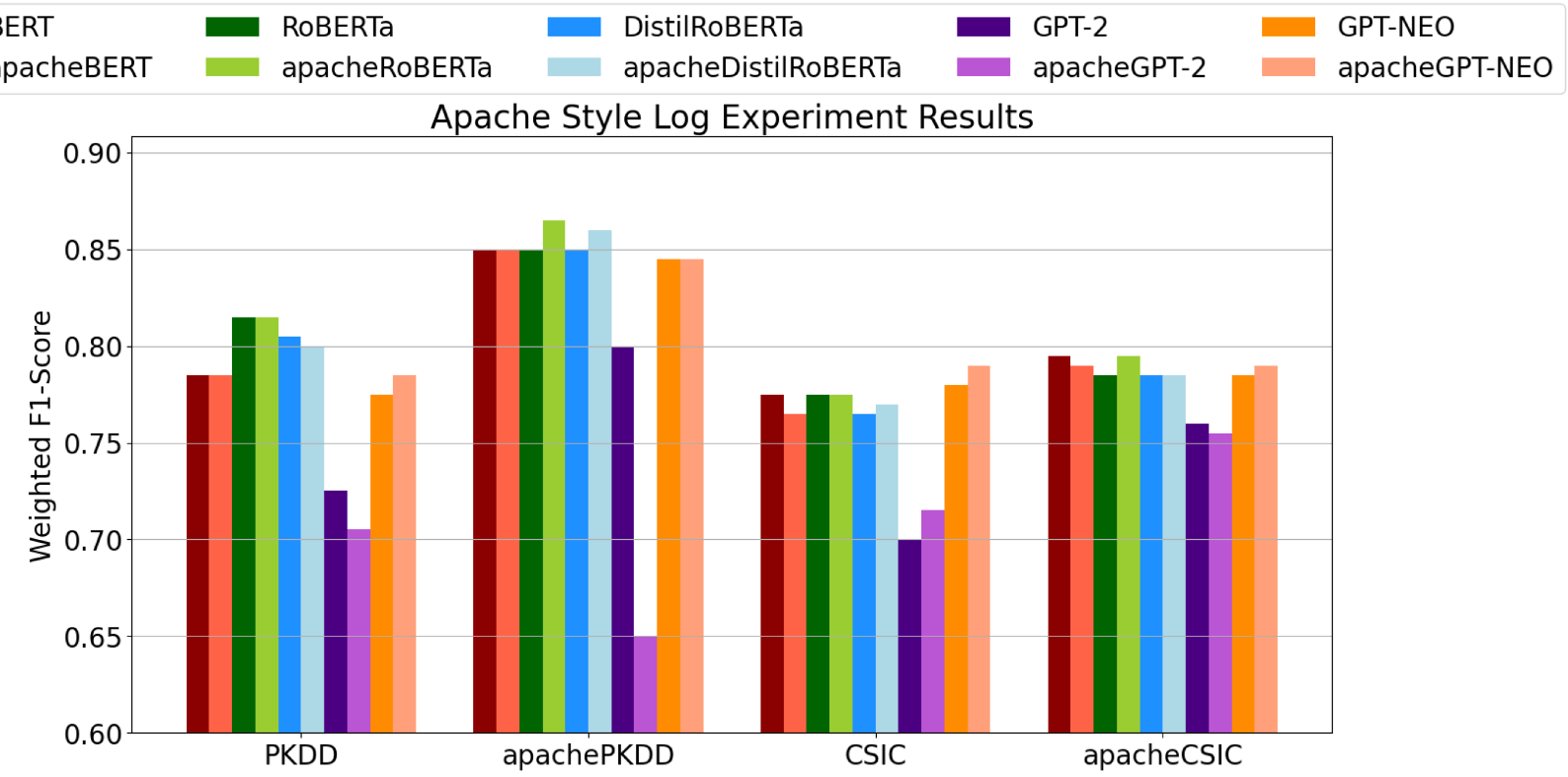

Figure 4.13: Pre-Processed Log Files Experiment Results: Weighted F1-Score

4.6.1 Results

In the case of the PKDD dataset, excluding the unique fine-tuned GPT-2 result, all learning model performance in anomaly detection was improved by 3.5-7.5%. All three BERT-style models performed at 0.85 for their baselines, their fine-tuned improved on this by 1-1.5%. The expected fine-tuning benefits were finally coming to light, in the case of the BERT-style language models, GPT-2 on the other hand lost performance yet again in the fine-tuning, but alongside GPT-NEO received a 5-7.5% boost in baseline performance with the restructuring. In the case of this anonymized dataset the language models were much better able to extract sentence embeddings that enabled disambiguation of anomalous events.

In the case of the CSIC dataset, all language model performance was increased by 1-6% with re-structuring of the logset. Baseline still outperformed on this dataset more than that of the fine-tuned models but clearly log structure was playing a larger role in the semantic feature extraction using these language models then was previously anticipated.

4.6.2 Discussion

After adapting the PKDD, and CSIC datasets to fit an Apache-style of log more similar to the fine-tuning log, and AA, an improvement is seen in the language models ability to extract meaningful sentence embeddings for anomaly detection. The up to 7.5% improvement seen in the PKDD dataset, and 6% in CSIC was unprecedented and indicates that there may be a way to continue improving the results if an optimal structure of log line can be ascertained. This improvement due to restructuring and removal of some features goes against the typical method of log analysis in which it can be assumed that the more features, and data you have the better the entropy is for identifying anomalous behavior. In the case of log analysis using semantic feature extraction, being able to make the log line tell a consistent story without as much noise in the form of some ubiquitous features provides improvement in anomaly detection.

Table 4.11: Re-structured Log Lines Experiment Results: Weighted F1-Score

	Weighted F1-Score					
	PKDD	apachePKDD	% Change	CSIC	apacheCSIC	% Change
bert-base-cased	0.785	0.850	+6.5%	0.775	0.795	+2%
RoBERTa	0.815	0.850	+3.5%	0.775	0.785	+1%
DistilRoBERTa-base	0.805	0.85	+4.5%	0.765	0.785	+2%
GPT-2	0.725	0.80	+7.5%	0.700	0.760	+6%
GPT-NEO	0.775	0.845	+7.0%	0.780	0.785	+0.5%
apacheBERT	0.785	0.850	+6.5%	0.765	0.790	+2.5%
apacheRoBERTa	0.815	0.865	+5%	0.775	0.795	+2%
apacheDistilRoBERTa	0.800	0.860	+6%	0.770	0.785	+1.5%
apacheGPT-2	0.705	0.65	-5%	0.715	0.755	+4%
apacheGPT-NEO	0.785	0.845	+6.0%	0.790	0.790	0.0%

4.7 Limitations

As previously outlined the semantic feature extraction is accomplished by utilizing the sentence embedding output produced by performing mean pooling on the output (final) layer of LLM. This presents a fixed length sentence embedding representation for any input sequence, in this case a log line. This sentence embedding representation is effectively the deep learning models interpretation of the input sequence and it represents it as a high dimensional vector representation. This vector representation is 768 float values between -1 and 1. This highly dense vector representation does not provide significant opportunity for human-interpretability. While there exists methods of improving interpretability like LIME, or SHAP [20], [34], they require that a supervised classifier be trained for sentiment analysis as a down stream task in the language model. Since this research focused on unsupervised methods of anomaly detection, and adaptation of the methodology for unsupervised algorithms would take significant time this method was not feasible to implement within the scope of this research. Thus, for the semantic feature extraction research, an emphasis was placed on interpretation of the t-SNE, and SOM visualizations as well as the performance of the unsupervised machine learning models trained on the featuresets. These methods provide adequate information on the language models abilities to extract meaningful sentence embedding representation for disambiguation of anomalous behavior in the security datasets.

4.8 Summary

In summary, this chapter provides an overview of the experiments performed in this study, the parameterization, results, and discussion thereof. The first set of experiments which explored the comparison of syntactic and semantic feature extraction techniques using the four datasets, and four different unsupervised machine learning approaches, identified that the semantic approach outperformed syntactic consistently, apart from in the case of the CSIC dataset where K-Means, and syntactic feature extraction saw better disambiguation of anomalous events. Semantic outperformed syntactic, albeit at a higher cost to feature extraction time, and model creation

times. The unsupervised machine learning algorithms saw SOM, HSOM, and K-Means perform the best consistently across both syntactic, and semantic techniques, though K-Means came with significantly reduced computation cost, particularly in comparison to HSOM. This set of experiment results led to two further questions, which continued the exploration of semantic feature extraction using LLMs with the following questions.

1. How do different LLMs impact the performance of semantic feature extraction-based log analysis and anomaly detection? Does fine-tuning improve it further?

2. How does the structure of the log line impact the success of the semantic feature extraction technique?

The second set of experiments explored the use of five different large language models for semantic feature extraction on the three application log datasets using K-Means for anomaly detection. This experiment also explored the impact of fine-tuning by individually fine-tuning each language model with 100,000 Apache log lines, and then evaluating the resulting language models semantic feature extraction. The results of this set of experiments indicate that DistilRoBERTa has the most consistent performance, and marginal improvement in fine-tuning, though all BERT-style models perform well in this experiment. More notably the two GPT models underperform in this benchmark, with remarkably high compuation cost for GPT-NEO with negligable improvement in performance, while GPT-2 underperforms consistently and also sees significant drop in performance after fine-tuning.

The third set of experiments explored the use of the same five different language models and the five fine-tuned variants while also re-structuring the security log data to match a Apache-style structure. This meant re-structuring PKDD, and CSIC to fit into a default Apache log line matching the form of the finetuning log data. This provided a 5-7.5% improvement in performance for the PKDD dataset, with 1-1.5% additional improvement in the fine-tuned models, and a 1-6% improvement in performance for the CSIC dataset using the language models, tough performance was still lost in the fine-tuned models. The boost provided by just re-structuring the log lines indicates opportunity for improved pre-processing of log lines for optimal

semantic feature extraction, since clearly the structure of the log line is impacting the success of semantic feature extraction method.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research has explored the efficacy of semantic and syntactic feature extraction techniques in combination with unsupervised machine learning algorithms across four different application log sets, as well benchmarking five different large language models for semantic analysis and anomaly detection. The research has provided compelling justification for NLP techniques to be adapted for use in log analysis. This is clearly seen in the case of the AA dataset, in which the learning models perform well on both the syntactic and semantic feature representations. By contrast the rigid structure of simplistic logs seen in ISOT lend themselves well to semantic feature extraction, and identify the limitations of the syntactic featuresets. Of the unsupervised machine learning algorithms exploration of SOMs provides a unique way of interpreting the featureset representations in the form of the hit map visualization. The SOM and HSOM models also yield improved performances in anomaly detection over K-Means, albeit at much higher computational cost. Alternately, agglomerative and isolation forest models in this research provided lower performance to that of K-Means. This exploration of syntactic and semantic feature representation of security log data led to two further questions which aimed to improve the semantic feature extraction methodology, since even though it outperformed the tf-idf based syntactic methodology in most cases it came at high computational cost.

This difference in performance for anomaly detection under different log representation is also explored with the domain adaptation of language models for log analysis. Evaluation of BERT, RoBERTa, DistilRoBERTa, GPT-2, and GPT-NEO on the three application logs identifies BERT-style architectural models as being more consistently capable of extracting sentence embedding representations which lend themselves to disambiguation of anomalous behavior. The performance of the largest model evaluated in this study, GPT-NEO brings into focus the limitations

of large models, in that the time required to perform semantic feature extraction is significantly higher than that of the smaller models, with little to no advantage in performance. This is obvious when the DistilRoBERTa model takes 4% of the time to extract features to that of the GPT-NEO and also outperforms it. The two GPT style models underperform in the task of semantic feature extraction log analysis, and continue to do so even after fine-tuning. The fine-tuning of the BERT-style models yeilds unconvincing improvement in generating meaningful log representations. Only providing less than 1% improvement in most cases. This changes when the log representation of the two hybrid logs CSIC, and PKDD is adapted and re-structured.

Re-structuring of the PKDD dataset in particular enables the language models to improve considerably in extracting meaningful semantic information, and this is reflected in the up to 6% improvement in F1-Score. The fine-tune and baseline improvement here with the apachePKDD, and apacheCSIC is more noticeable indicating that the fine-tuning process does enable slightly improved semantic feature extraction when the input sequence more closely resembles the dataset used in fine-tuning.

5.2 Future Work

Given the results of this research, in order to further improve upon the area of NLP-based feature extraction in log analysis more exploration of LLM based semantic feature extraction is necessary. This study was able to identify some of the limitations of current state of the art language models in log analysis, which include high computation times, a link between log structure and performance as well as unimpressive fine-tuning results. Future work should also explore smaller models for improved feature extraction times, and optimization of log representation. On the syntactic side, the results of the small feature representation with only seven features indicate that even with minimal computation time, and a much smaller representation, disambiguation of abnormal behavior can still be performed within 10-15% of the much more computationally heavy semantic methods. Exploration of new syntactic characteristics to add to this syntactic representation may improve the entropy of the features selected and could potentially close the gap in performance.

Bibliography

[1] Akiko Aizawa. An information theoretic perspective of tf-idf measures. *Information Processing Management*, 39:45–65, 01 2003.

[2] E. Alpaydin. *Introduction to Machine Learning, second edition*. Adaptive Computation and Machine Learning series. MIT Press, 2009.

[3] G. H. Ball and David J. Hall. Isodata, a novel method of data analysis and pattern classification. 1965.

[4] Mansi Bhatnagar, Gregor Rozinaj, and Puneet Kumar Yadav. Web intrusion classification system using machine learning approaches. In *2022 International Symposium ELMAR*, pages 57–60, 2022.

[5] Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. If you use this software, please cite it using these metadata.

[6] Matteo Boffa, Giulia Milan, Luca Vassio, Idilio Drago, Marco Mellia, and Zied Ben Houidi. Towards nlp-based processing of honeypot logs. pages 314–321, 2022.

[7] Rafael Copstein, Egil Karlsen, Jeff Schwartzentruber, Nur Zincir-Heywood, and Malcolm Heywood. Exploring syntactical features for anomaly detection in application logs. *it - Information Technology*, 64(1-2):15–27, 2022.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[9] Michael Dittenbach, Dieter Merkl, and Andreas Rauber. Growing hierarchical self-organizing map. volume 6, pages 15 – 19 vol.6, 02 2000.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.

[11] ECML/PKDD. Ecml/pkdd 2007 discovery challenge, September 2021. https://gitlab.fing.edu.uy/gsi/web-application-attacks-datasets/-/tree/master/ecml_pkdd.

[12] Brian Gallagher and Tina Eliassi-Rad. Classification of http attacks: A study on the ecml/pkdd 2007 discovery challenge. 01 2009.

[13] Carmen T Giménez, Alejandro P Villegas, and Gonzalo Á. Marañón. HTTP Dataset CSIC 2010, 2010.

[14] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. pages 1–8, 2021.

[15] Muhammad Anis Al Hilmi, Kurnia Adi Cahyanto, and Muhamad Mustamiin. Apache web server - access log pre-processing for web intrusion detection, 2020.

[16] Gunes Kayacık, A. Zincir-Heywood, and Malcolm Heywood. A hierarchical som-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, 20:439–451, 06 2007.

[17] Gunes Kayacık, A. Zincir-Heywood, and Malcolm Heywood. A hierarchical som-based intrusion detection system. *Engineering Applications of Artificial Intelligence*, 20:439–451, 06 2007.

[18] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[19] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojarvi, Jukka Honkela, Vesa Paatero, and Antti Saarela. Self organization of a massive document collection. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 11:574–85, 02 2000.

[20] Enja Kokalj, Blaž Škrlj, Nada Lavrač, Senja Pollak, and Marko Robnik-Šikonja. BERT meets Shapley: Extending SHAP explanations to transformer-based classifiers. In *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, pages 16–21, Online, April 2021. Association for Computational Linguistics.

[21] Md Tahmid Rahman Laskar, Jimmy Xiangji Huang, Vladan Smetana, Chris Stewart, Kees Pouw, Aijun An, Stephen Chan, and Lei Liu. Extending isolation forest for anomaly detection in big data via k-means. *ACM Trans. Cyber-Phys. Syst.*, 5(4), sep 2021.

[22] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. pages 413–422, 2008.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[24] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[25] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[26] Ali Moradi Vartouni, Mohammad Teshnehlab, and Saeed Sedighian Kashi. Leveraging deep neural networks for anomaly-based web application firewall. *IET Information Security*, 13(4):352–361, 2019.

[27] Sukhyun Nam, Jae-Hyoung Yoo, and James Won-Ki Hong. Vm failure prediction with log analysis using bert-cnn model. pages 331–337, 2022.

[28] Hai Thanh Nguyen and Katrin Franke. Adaptive intrusion detection system via online machine learning. In *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, pages 271–277, 2012.

[29] Harold Ott, Jasmin Bogatinovski, Alexander Acker, Sasho Nedelkoski, and Odej Kao. Robust and transferable anomaly detection in log data using pre-trained language models. *CoRR*, abs/2102.11570, 2021.

[30] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[31] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019.

[32] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[33] Suseela Sarasamma, Qiuming Zhu, and Julie Huff. Hierarchical kohonenen net for anomaly detection in network security. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 35:302–12, 05 2005.

[34] Mateusz Szczepański, Marek Pawlicki, Rafał Kozik, and Michał Choraś. New explainability method for bert-based model in fake news detection. *Scientific Reports*, 11, 12 2021.

[35] University of Victoria. Isot-cid cloud security, October 2021. https://www.uvic.ca/ecs/ece/isot/datasets/cloud-security/index.php?utm_medium=redirect&utm_source=/engineering/ece/isot/datasets/cloud-security/index.php&utm_campaign=redirect-usage.

[36] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[37] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[39] Farzin Zaker. Online Shopping Store - Web Server Logs, 2019.

[40] Jing Zhang, Zezhou Li, Xianbo Zhang, Feng Lin, Chao Wang, and Xingye Cai. Posbert: Log classification via modified bert based on part-of-speech weight. pages 979–983, 2022.

[41] Ying Zhao, George Karypis, and Usama Fayyad. Hierarchical clustering algorithms for document datasets. *Data Min. Knowl. Discov.*, 10:141–168, 03 2005.

Appendix A

Appendix

A.1 Appendix A

Table A.1: Semantic vs Syntactic vs Bi-Gram Feature Extraction Times (Seconds)

	Feature Extraction Time (Seconds)			
Extraction Technique	AA	ISOT	CSIC	PKDD
Semantic	44	60	138	217
Syntactic	5	4	11	27
Character Bi-Gram	1393	1534	3961	5688

Table A.2: Model creation time for character Bi-Gram featureset

	Model Training Time (Seconds)			
Unsupervised ML	AA	ISOT	CSIC	PKDD
Isolation Forest	18,576	21,452	21,894	24,341
Agglomerative	21,561	25,823	27,950	29,411
K-Means	6,381	7,707	8,103	8,984
SOM	54,454	78,301	85,801	102,351
HSOM	81,163	115,430	127,412	155,708

Table A.3: Model performance for character Bi-Gram featureset

	Weighted F1-Score			
Unsupervised ML	AA	ISOT	CSIC	PKDD
Isolation Forest	0.78	0.56	0.64	0.58
Agglomerative	0.66	0.52	0.58	0.48
K-Means	0.81	0.76	0.73	0.68
SOM	0.91	0.85	0.83	0.78
HSOM	0.93	0.87	0.83	0.80

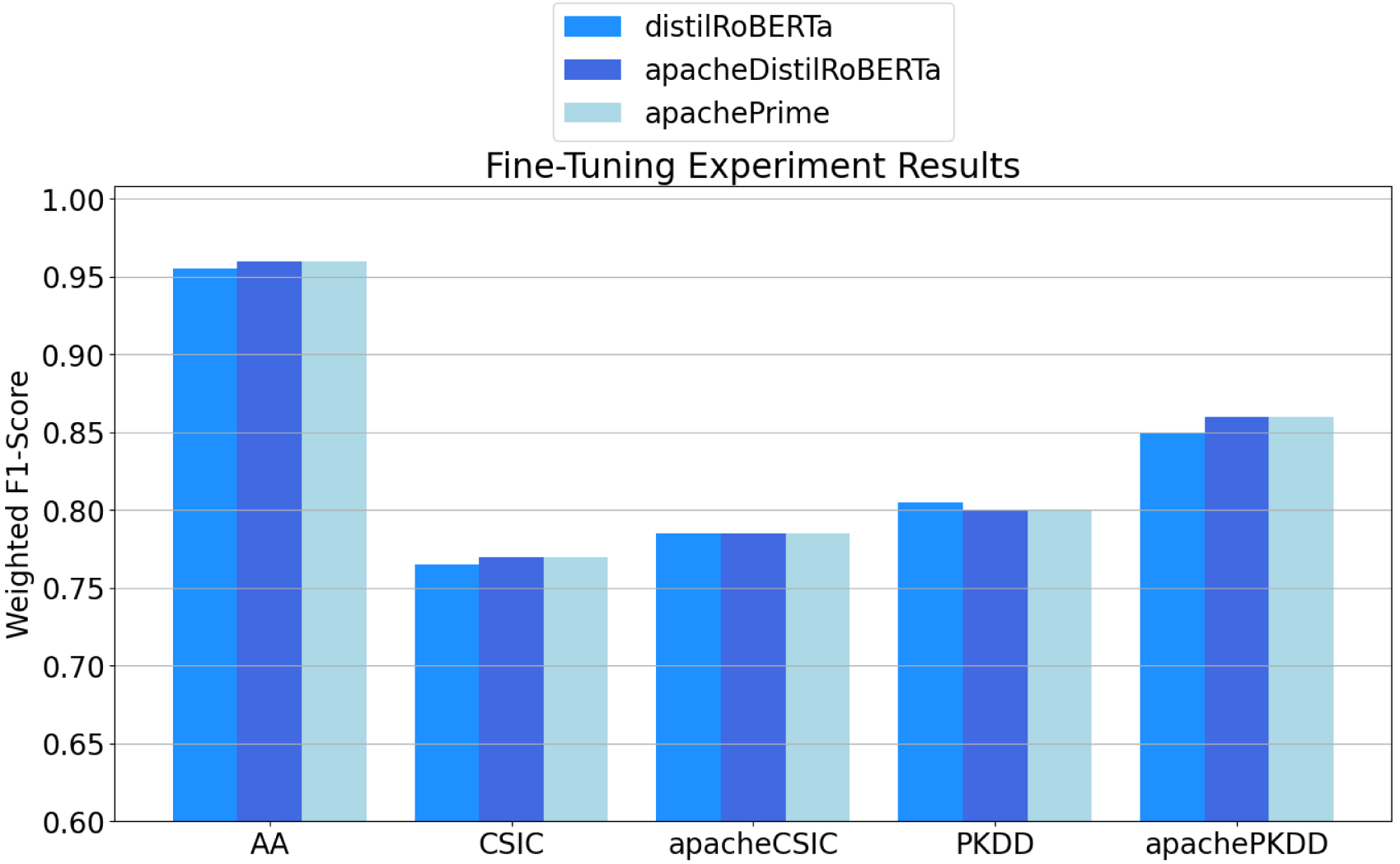

Figure A.1: distilRoBERTa, 100,000 samples fine-tuned distilRoBERTa (apacheDistilRoBERTa), and 200,000 samples fine-tuned distilRoBERTa (apachePrime)