CHALMERS

Critical Event Prediction in Logs at Customer Network

Predicting Next Critical Alarm using Multimodal Logs and Natural Language Processing Techniques

Master's thesis in Mathematics

ELMAR HAJIZADA

Department OF Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022

Master's thesis 2022

Critical Event Prediction in Logs at Customer Network

Predicting Next Critical Alarm using Multimodal Logs and Natural Language Processing Techniques

ELMAR HAJIZADA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department OF Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2022

Critical Event Prediction in Logs at Customer Network
Predicting Next Critical Alarm using Multimodal Logs and Natural Language Processing Techniques
ELMAR HAJIZADA

Supervisor: Johan Jonasson, Department of Mathematical Sciences
Rozita Akrami, Ericsson AB
Examiner: Johan Jonasson, Department of Mathematical Sciences

Master's Thesis 2022
Department OF Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Critical Event Prediction in Logs at Customer Network
Predicting Next Critical Alarm using Multimodal Logs and Natural Language Processing Techniques
ELMAR HAJIZADA
Department OF Mathematical Sciences
Chalmers University of Technology

Abstract

Implementing effective maintenance prognosis for Radio units at Ericsson can result in a number of benefits, including better system safety, improved operational reliability, longer equipment lifespan, and lower maintenance costs. Preventive investigations and repairs on the hardware and software level can be done to avoid the radio unit from failing by forecasting whether or not the radio unit will have an alarm in the near future. The goal of this thesis was to use multiple logs taken from a radio unit to predict whether an alarm would occur in the next one to nine days. The log file contents have been divided into chunks using different approaches like expanding window, independent chunks and time interval chunks where each chunk labeled according to timestamp of the alarm. Ericsson has used a combination of verdicts (features that are defined by subject matter experts) to extract the best features from the log files. This rule-based approach is inefficient since it requires modification of the script using expert knowledge when there is a change in the design of the hardware.

The purpose of this thesis project was achieved using data-driven NLP approaches including log parsers and word embeddings. An independent chunks approach with Drain log parser using concatenated bag-of-words representations for each log file fitted on the Xgboost model outperformed other combination of log parsers and word embeddings. LSTM model was used with 1 day interval chunks to see if the complex sequential model can achieve a sufficient score. Experiments using complex sequential model, such as the LSTM many-to-many model with doc2vec embedding, have shown shown that they can predict alerts before they occur. All the tested models were evaluated using cross-validation. The Xgboost model with the independent chunks approach using Drain log parser and BOW embedding achieved an average F1-score of 0.873, LSTM model with time interval chunks approach using doc2vec embedding achieved average 0.853 F1-score across shifting time periods from one to nine days.

Keywords: NLP, log, predictive maintenance, classification, machine learning, word embedding, LSTM, XGBoost, AWSOM-LP, Drain.

Acknowledgements

I'd want to express my gratitude to Ericsson for providing me with the chance and resources to complete this thesis. I'd want to express my gratitude to Rozita Akrami, my Ericsson supervisor, who was personally responsible for initiating this thesis project with me. I'd like to express my gratitude to Johan Jonasson, my supervisor and examiner at the Department of Mathematical Sciences, for providing me with guidance and valuable input on the thesis' theories and methods.

ELMAR HAJIZADA, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

The major data source for system, application or network observability is log files. The log files' availability aids developers in troubleshooting and debugging existing issues within the device. The number of log files created by devices grows and becomes more complex to debug, as the system scales up. When devices malfunctions, manually debugging and resolving the problem becomes practically impossible.

Domain expertise with a deeper understanding of the process assist developers in this regard to eliminate the requirement for manual debugging on the log files. At Ericsson such logs are used for debugging and analyzing source of the fault for particular unit that has returned from customer network. If there is a hardware problem, the unit can be sent to a repair center; however, if there is only a software problem, or if there is no problem at all, the sending the unit to repair center can be avoided. Detaching the units from the customer network itself is costly process for Ericsson. As a result, there is a need to detect if a unit in a customer network is faulty, or even to be proactive and predict when a failure will occur. To identify the fault, the screening system now uses a combination of verdicts (features that are defined by subject matter experts) and the output of a rule-based script on logs collected from the unit. However, this system is not very efficient, and the rule-based approach requires modification of the script using expert knowledge when there is a change in the design of the hardware.

To mitigate this, ML models have been trained on these historical logs in a supervised fashion to aid in fault detection. While this is a helpful aid to the SMEs, it still requires the entirety of the logs in the returned units. It can, therefore, only be performed after the unit is transported to the screening center. Still, on the other hand, some of the logs in the units are written continuously during the operation in the network. It could perhaps be possible to identify problems or critical events exactly when, or even before they occur based on the events in these logs.

1.1 Problem Statement

Several critical events may occur before the device finally breaks down and becomes faulty. Raised alarms in the customer network can be shown as an example of critical events. While the device is connected to the customer's network, a variety of issues such as temperature, power, configurations, and so on may develop, resulting in appropriate alarms being triggered.

The goal of this thesis project is to investigate methods for developing a proactive model that can forecast whether or not an alert will be triggered several days in advance. As an extended goal, predicting type of the alarm that is going to be triggered will be developed. To develop this project, data-driven approaches such as Natural Language Processing techniques will be examined in order to lessen dependency on SMEs. Various log parsers such as Drain and AwsomLP with NLP techniques will be benchmarked to get a comprehensive understanding of proactive logs and their data-driven features.

1.2 Dataset

A suitable dataset is required to investigate and develop a ML model to accomplish this task. At Ericsson, while a device is connected to a customer's network, several log files such as **elog**, **hwlog**, and **alarm log** are generated. Elog is written in a streaming fashion, available both when an alarm is triggered as well as if a unit is returned. Hwlog is available both when an alarm is triggered as well as if a unit is returned. Finally, alarm log is written in a streaming fashion and is only available when an alarm is triggered and the unit is still live at the customer network. Alarm log will be used to annotate the elog, and hwlog lines based on time, when the alarm triggered.
Alarms can be triggered while logs are being generated in the background of a specific radio unit, and the system will automatically reboot or restart to force the unit to a non-faulty state. If restart or reboot operations are not successful then the radio unit will be taken to the repair center and detached from customer network. In a lifetime of radio unit at customer network, several alarms of different types can be triggered in different timestamps. Overview of the dataset can be seen in Table 1.1
As can be seen from Table 1.1, each serial number (showing radio product ID) can have several alarms and timestamps for each alarm.

Table 1.1: Snippet from dataset

Serial	Timestamps	Alarms
E553405312	2020-12-24 06:08:23	HW Partial Fault
E553164758	2021-05-04 02:57:27, 2021-05-04 03:08:12	No Connection SW Error
E552852522	2020-06-24 06:49:56, 2021-05-04 03:08:12	VSWR Over Threshold Inconsistent Configuration

In this thesis period, specific radio product named as **Radio2219** dataset has been used to investigate possible methods and achieve expected goals. There are possible 8 failure types for **Radio2219** dataset and distribution of those failure types has been provided in Table 1.2. As can be seen from the table 1.2, **Rf Reflected Power High** and **Feature Resource Missing** are the least frequent failure types in our

dataset. These failure types has been removed from the dataset since there are not enough amount of occurences to evaulate them.

As previously stated, the purpose of the thesis is to develop a proactive model that can identify many alarms in a sequential, streaming manner for each radio unit in the customer network. Different log formats hold information about certain sections of the radio units that may give insights to determine the fault type. To classify failure type, 3 log types that listed above will be used.

Table 1.2: Distribution of failure types in Radio2219 dataset

Failure type	**Frequency**
SW Error	455
HW Partial Fault	413
Inconsistent Configuration	218
No Connection	186
VSWR Over Threshold	139
Power Loss	76
Current Too High	51
HW Fault	30
Rf Reflected Power High	12
Feature Resource Missing	5

These alarm labels can be separated into two types as shown in Table 1.3, based on discussions with subject matter experts: DU (digital unit) Raised Alarms and Radio Fault Indications.

Table 1.3: Grouping failure types into 2 categories

DU Raised Alarms	**Radio Fault Indications**
SW Error	HW Fault
VSWR Over Threshold	HW Partial Fault
Inconsistent Configuration	Current Too High
No Connection	Power Loss

1.3 Thesis scope and limitations

The greatest limitation in this thesis project is that there is not any rule-based approach or data-driven model provided by Ericsson regarding to this thesis project. The data-driven proactive model that is built throughout the project will be the first one used inside the team and will be used as a benchmark for future developments around this project.

1.4 Thesis Outline

The background of the problem and associated work will be discussed in the second chapter. The theory part of the strategies employed throughout the thesis time will be covered in the third chapter. The fourth chapter will discuss the methods that were employed to achieve the project's objectives. The next chapters will include the results, discussion, and conclusion, in that order.

2

Background

2.1 Log Analysing

The process of evaluating and understanding computer-generated documents known as logs is known as log analysis. A variety of programmable technologies, including as networking devices, operating systems, and applications, produce logs. A log is a collection of messages in chronological order that describe activities occurring within a system. Log files can be broadcast to a log collector over an active network or saved in files to be reviewed later. In any case, log analysis is the careful process of evaluating and interpreting these messages in order to obtain insight into the system's inner workings.

System behavior, faults, and future problems can all be deciphered through log analysis. To find the core cause of an anomaly or malfunction, system domain experts frequently must manually comb through log files and repeatedly filter out and organize log items. More effective diagnostic methods might thereby reduce some of the expense of this downtime and several ways have been investigated in the field of computer systems diagnostics. Log files are usually available in large quantities, although their appearance varies greatly depending on the system. Domain experts frequently assist developers in developing a rule-based system that can aid in extracting additional information from log files. Instead of manually searching log files for patterns, domain experts can use these created algorithms to extract useful information and comprehend system behavior.

Machine learning techniques may be a potential answer to the problem of reducing the reliance on domain knowledge. Traditional supervised machine learning relies on sufficient and well classified dataset, in addition to some domain expertise for feature extraction. It is frequently difficult to acquire such information from log files. Going through log files and identifying relevant features is a time-consuming procedure for domain specialists. Once the structure of log files changes, domain experts need to go through log files again to ensure that they are not missing any vital information. Furthermore, relying on domain specialists may lead to the omission of crucial characteristics that they are unaware of. As a result, data-driven methods can be utilized to eliminate the need for subject matter experts.

2.2 Natural Language Processing

Natural language processing (NLP) is an area of computer science concerned with computers' capacity to grasp text and spoken language in the same way that humans can. When it comes to textual data, NLP is quite significant. NLP allows computers to do a wide variety of natural language activities at various levels, from parsing and part-of-speech (POS) tagging to machine translation and chat systems. Machine learning methods to NLP problems have relied on shallow models trained on very high-dimensional and sparse features for decades. In recent years, neural networks based on dense vector representations have excelled at a variety of NLP tasks. The success of word embeddings [3], and deep learning approaches [2] has fueled this trend. In numerous NLP tasks, such as named-entity recognition, semantic role labeling, and part-of-speech tagging, the paper [4] showed that a simple deep learning framework performs better than most state-of-the-art approaches. The authors stated that they attempted to avoid task-specific engineering by ignoring a significant amount of prior knowledge. The system learns internal representations using enormous amounts of mostly unlabeled training data, rather than relying on highly tailored man-made input attributes for each task. This approach is then used to create a free tagging system with low computing requirements.

The use of NLP with textual data can include log analysis, log mining, and anomaly identification. Recent research have shown that NLP techniques can be quite useful in log analysis. Identifying anomalies or classifying fault types are two examples. This article [1] investigates log mining with Natural Language Processing and its application to anomaly detection. To achieve high accuracy in spotting anomalies, the authors used multiple NLP techniques such as word2vec, Bag-of-words.

For detecting anomalies in log data, [5], [6] the most prevalent use case has been to use sequential neural networks such as Recurrent Neural Networks (RNN) or Long-Short-Term-Memory models. The models' sequential qualities enable them to detect patterns in normal activity as reflected in log data, as well as react when the pattern is violated. This allows for binary classification of whether the logs are demonstrating normal or abnormal behavior, as well as the detection of when these potential anomalies occur. The multi-label classification of logs is a related issue that has received less attention. Instead of determining whether or not a log is unusual, the model should be able to determine the type of problem or fault in the system.

With the release of the 'Attention Is All You Need' paper[7], transformer-based architectures have redefined the field of Natural Language Processing (NLP) and established the state of the art for a variety of AI benchmarks and tasks. On NLP-related tasks like machine translation, the Transformer model beat many prominent sequential models with lower temporal complexity. The paper had a significant impact on NLP research, resulting in numerous new articles and high-performing language models that demonstrated new state-of-the-art performance [8]. By focusing on specific parts of the source text during translation, an attention mechanism has recently been used to improve neural machine translation. Paper[7] demon-

strates that the transformer-based attention mechanism surpasses the best previously reported models (including ensembles) by more than 2.0 BLEU, achieving a new state-of-the-art BLEU score.

2.3 Related Work

Since the focus of thesis is to predict the failure before it happens, different papers regarding to the topic of predictive maintenance have been researched. Event log prediction is a well-studied issue with numerous applications. In one of the papers [9], authors try to solve the problem of event prediction in aviation using collected event loggings. The goal is to create an alerting system that will alert aviation engineers well in advance of impending aircraft faults, giving them adequate time to plan for the necessary maintenance. That paper used a regression approach to anticipate the next failure event that could occur. Anomaly detection based on logs is also commonly utilized in industrial systems. Authors provide a data-driven way to having scheduled corrective maintenance operations in this study [10]. They attempted to calculate the probability of failure within the given time frame. To get the most out of log messages and event logs, machine learning approaches, data mining, and feature extraction techniques were applied. The authors conclude that machine failures can be accurately predicted up to 168 hours ahead of time.

The paper [11] shows how fault-tolerant data stream processing can benefit from online failure prediction. Based on collected feature streams, authors use stream-based online learning algorithms to continually identify runtime operator status as normal, alert, or failure. IBM System stream has been used for processing system to create the online failure forecast system. From the tests authors show that the online failure forecast system can predict a wide range of stream processing software faults with high accuracy while imposing minimal cost on the stream system. To achieve online prediction, the authors leverage software sensors and system logs, as well as a decision tree model on top of them.

Paper [12] presents a deep learning approach to automate system failure prediction. Paper describe a novel approach for predicting IT system failures by automatically parsing streamed console events and detecting early warning signs. The approach, in particular, contains a log template extraction mechanism that clusters logs with similar structure and content. By interpreting each pattern as a word and the set of patterns in each discretized epoch as a document, we resemble the TFIDF notion. This results in a feature space with a much lower dimensionality that can offer reliable system status signals. Recurrent neural network, called Long Short-Term Memory (LSTM), has been used to deal with the "rareness" of labeled data in the training phase because system failures are uncommon. The main part of this approach is that authors use log pattern learning using clustering, pattern recognition and pattern matching techniques and produce log clusters and log patterns. Pattern based tf-idf feature extraction has been used to derive patterns from log clusters and log patterns.

Failure prediction has been tackled by statistical methods in addition to classification-based approaches. The authors of [13] present a different strategy based on survival analysis. They employ the COX proportional hazard model [14] to approximate the survival function, and the occurrence of other events/failures is used as a covariate (features). A prediction is made in this setup by predicting the interval during which a failure is likely to occur. Furthermore, the authors suggest in [15] a method for learning patterns from event log streams and predicting failures by calculating the chance of a failure occurring given the patterns found.

3
Theory

3.1 Log Parsing

Usually log files that are unstructured and it becomes difficult to analyze and comprehend their structure without transforming them to a structured format. These methods are used to transform unstructured raw log events into a structured format for future analysis and break down long log messages into structured patterns. For instance, one way of breaking down a log message could be dividing it into seperate parts such as ip address, timestamp, log message content and etc. By doing this we can achive a structured format and understand the data better.
Machine learning models can use log parsing methods as a preprocessing step. These techniques also aid in the detection of dynamic variables and static patterns in log messages. Template structure is common in log messages, although they also contain certain dynamic variables. As the table 3.1 shows, two sentences have nearly identical structures with a few minor changes. Only some words distinguish these two messages, which are otherwise identical. These words are dynamic variables, while the rest of the tokens are static variables.

Table 3.1: Overview of dynamic and static parameters

Log message	**Log template**	**parameter**
Restart cause:COLD; LMC ID: CXP901096	Restart cause <*> LMC ID<*>	COLD, CXP901096
Restart cause:ORDERED; LMC ID: CXP90173	Restart cause <*> LMC ID<*>	Ordered, CXP90173

One of the simplest parsing methods is to define regular expressions with manual analysing in order to filter out unnecessary information from log messages. The major downside of this strategy is that new log templates may appear in the future, necessitating the identification of new regular expressions. There are various log parsing methods that automatically parse the logs. Although the main log message structure should be specified, dynamic and static terms will be automatically detected. In the recent years there has been several log parser methods developed and published such as Drain (An Online Log Parsing Approach with Fixed Depth Tree) [16] and AwsomLP (An Effective Log Parsing Technique Using Pattern Recognition and Frequency Analysis) [17]. These methods are used in this thesis to identify dynamic, static words and cluster the log templates. In total, there are 3 log parsing

methods that have been used in this thesis project: simple regular expression parser, drain parser and awsomlp parser.

3.1.1 Drain

Drain is a fixed depth tree based online log parsing method [16]. This parsing method helps to automatically transform unstructured log templates into structured one. The parser method consist of several stages.

First step is to use domain knowledge and preprocess the log templates since these processing steps can help to improve the accuracy of the log parsing. Using simple regular expressions several commonly-used variables such as IP addresses can be removed from the templates.

Second step is to build and traverse the *Parser Tree* as shown in Figure 3.1.1, to find appropriate leaf node which represents log message template. Drain starts from a root node and traverse each tree level. The first level nodes represents the number of words in log template. When a log entry is received, it is initially matched to one of the root node's children with the same amount of tokens as the message. If no such node exists that matches the number of words in a new log message, a new node with the **number of words** in the new log message is generated.

Once first level node is identified, the next step is to match the log messages to templates based on the preceding tokens. If the first token of the log messages matches to the token in the node then that node is selected and traversing will continue until it reaches to the leaf node. If no such node exists, a new node with the first token of the log message will be created.

Once the drain has reached to the leaf node, which can contain a lot of log groups, the last step will be to select the most suitable group based on similarity measurements. *SimSeq* is similarity measurement that drain uses to compare similarities between log message and log template of each log group. The formula is defined as follows:

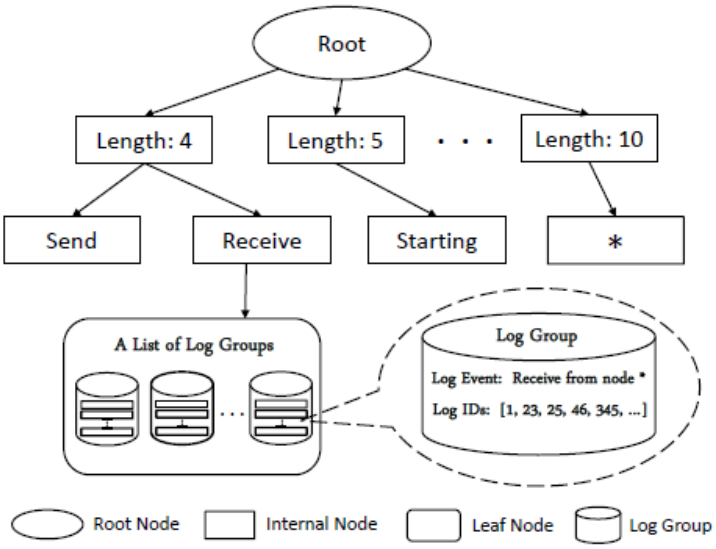
$$SimSeq = \frac{\sum_{i=1}^{n} equ(seq_1(i), seq_2(i))}{n} \tag{3.1}$$

where seq_1 and seq_2 represents two log messages and n shows number of tokens in log entry. Equ functions is defined as follows:

$$equ(t_1, t_2) = \begin{cases} 1, & \text{if } t_1 = t_2 \\ 0, & \text{otherwise} \end{cases}$$

Last step is to update the parsing tree which can be done by adding log ID of log message to the log ID list of identified group. If drain can not find a suitable group then, it creates a new group with new log ID and log template. The general structure of Drain parser algorithm can be seen from the following figure:

Figure 3.1: Structure of Drain Parser



3.1.2 Awsom-LP

Awsom-LP is an effective log parsing technique using pattern recognition and frequency analysis [17]. The parser has three main stages:

- The first step of the algorithm is to pre-process the log messages using regular expressions. Header information such timestamp, log level and etc will be removed from the log messages according to the parser method. Some dynamic variables such as IP address, timestamps can be replaced by wildcard symbol to improve the accuracy of the parser

- Second step is to group similar log messages into same templates, which will be used to seperate dynamic and static tokens. Frequency analysis will be used to distinguish dynamic and static variables. Grouping strategy is based on simple string matching. Similarity between two log message L1 and L2 is counting number of letters that are not numbers or delimiters (commas, semicolon, quotes and etc) in L1 and L2. The ratio of these two numbers will be similarity ratio. However, threshold value should be used to decide if two log message is similar or not. Authors use 100% as a threshold value in the paper. The formula for similarity measurement is as follows:

$$similarity(L_1, L_2) = \frac{count(L_1)}{count(L_2)} \tag{3.2}$$

  As an example, in below table 3.2 one can see several log messages that belongs to the same cluster group based on similarity measurement where threshold is 100%:

Table 3.2: Log messages that belongs same cluster based on similarity measurement

Log messages from same cluster
PacketResponder 1 for block *blk*_38865049064139660 terminating
PacketResponder 0 for block *blk*_6952295868487656571 terminating
PacketResponder 2 for block *blk*_8229193803249955061 terminating
PacketResponder 2 for block *blk*_6670958622368987959 terminating
PacketResponder 2 for block *blk*_572492839287299681 terminating

- To discriminate between static and dynamic tokens, frequency analysis is applied to the instances of each pattern that were found in the previous stage. This is done by counting the number of times each term appears in all instances of the same pattern. To separate dynamic and static tokens, authors utilize the lowest possible frequency, which is usually 1. To put it another way, every token that appears more frequently than the minimal frequency is considered a static token. Authors are aware that doing so may cause some dynamic tokens to be misclassified as static. Finding a threshold that provides the optimal trade-off is difficult. So, as a simplicity authors use the minimum frequency in the cluster as a threshold value. In the below figure the minimum frequence is considered as 1, so all the tokens that has frequency more than this threshold will be known as static tokens and dynamic tokens are replaced by wildcard symbol. The table 3.3 shows the frequency of each entry and type of the token after comparing to the minimum frequency

Table 3.3: Log messages that belongs same cluster based on similarity whe measurement where minimum frequency threshold is 1

Token	**Frequency**	**Parameter type**
PacketResponder	5	static
0	1	dynamic
1	1	dynamic
2	3	static
for	5	static
block	5	static
blk_38865049064139660	1	dynamic
blk_6952295868487656571	1	dynamic
blk_8229193803249955061	1	dynamic
blk_6670958622368987959	1	dynamic
blk_572492839287299681	1	dynamic
terminating	5	static

- The last step of Awsom-LP parser is to post-process the log templates. As a post-processing step authors remove the numerical dynamic tokens which were not identified in the previous step. The numerical variables that appear between brackets, parentheses and spaces are removed from the log messages.

3.2 NLP Techniques

3.2.1 TF-IDF

The TF-IDF statistic investigates the relevance of a word to a document in a set of documents. Numbers are required as input for machine learning models. It's critical to convert textual data into numbers when interacting with it. Text vectorization is the process of turning text into numbers, and tf-idf is one of the most widely used vectorization algorithms when working with textual data. Each document is represented by a vector, with each vector index representing a single word and the value of the corresponding vector index denoting the word's TF-IDF value. This is done by simply multiplying two metrics: the number of times a word appears in a document and the word's inverse document frequency over a set of documents. However, it should be pointed out that word orders are not taken into account in this method.

- The term frequency of the document is calculated by measuring how many number of times word appear in an entire document. The formula for term frequency is as follows:

$$TF(t, d) = \frac{f_{t,d}}{|d|} \tag{3.3}$$

  where $f_{t,d}$ represents how many times word t appear in document d and $|d|$ shows number of words in the document d.

- The inverse document frequency is a metric that helps to know how common is the word appears in entire document. The *IDF* metric is calculated as follows:

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3.4}$$

  Where |D| represents number of documents in the set and denominator shows number of documents that contains word t in it.
- Final value for term t of a document d in a document set d is multiplication of these two metrics:

$$TF\text{-}IDF(t,\ d) = TF(t, d) * IDF(t, D) \tag{3.5}$$

3.2.2 Bag-of-Words

The Bag-of-Words model is another vectorization method that used to convert textual data into numbers. Simply said this method is about counting the occurrence of each word in document. This vectorization approach generates a vector for each text, with each vector index representing a single word and the value of the corresponding vector index indicating the word's occurrence value. We only count words and ignore grammatical characteristics and word order. Any information about the

sequence or structure of words in a document is discarded, which is why it's considered a "bag" of words. The model just looks at whether or not recognized terms appear in the document, not where they appear. In order to find BoW value for a word in a document, it is required to know the vocabulary in entire set of documents.

Table 3.4: Log samples from elog and BoW model

Vocabulary / Documents	RL	Throttling	started	all	command	elog	COLI	Counters
RL Counters all command	1	0	0	1	1	0	0	1
Throttling started elog started	0	1	2	0	0	1	0	0
COLI Counters command all	0	0	0	1	1	0	1	1

As can be seen from Table 3.4, we have set of 3 documents and vocabulary where size |N| is 8. Each row contains occurance of the word in particular row document.

3.2.3 N-Grams

Text N-grams are commonly employed in text mining and natural language processing. They're essentially a collection of co-occurring words within a specific frame. N specifies the number of words in a sequence we want to use as a feature. Because the TF-IDF and BoW models do not provide any sequential information about words in documents, ngrams can be utilized to learn more about word ordering. As an n-grams, the following example can be shown:

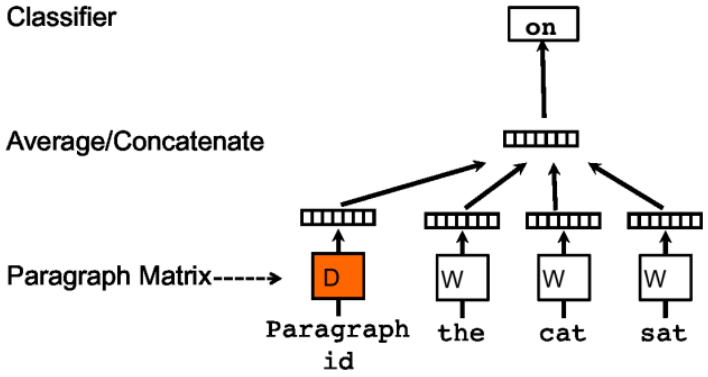
Table 3.5: N-grams for the log message from elog

Log message	unigram	bigram	trigram
COLI Counters command all	'Coli' 'Counters' 'command' 'all'	'Coli Counters' 'Counters command' 'command all'	'COLI Counters command' 'Counters command all'

3.2.4 Doc2Vec

Machine learning models usually require fixed-length input vector. One example is Bag-of-Words which provides fixed input vector for the machine learning models. The major drawback of the BoW model is that it does not consider word ordering or semantics of the text paragraph. There are other word embeddings models like word2vec [3] which helps to get relationships between words. However, word2vec model outputs vectors for each word and it does not consider semantic relationship between documents. In this thesis, semantic relationships between documents was taken into account. Therefore, Doc2Vec [21] representation was the main focus since it takes semantic relationships between document into consideration. Previous method [3] that has been used to output word vectors is the main inspiration while obtaining document to vector embeddings [21]. Word vectors are asked to predict the target word given the sorrounding words. Paragraph vector also utilizes this idea, trying to predict the target word given many contexts sampled from the paragraph.

Figure 3.2: A framework for learning paragraph vector: Distributed memory model [21]



3.2.4.1 Paragraph Vector: Distributed memory model

In this method, every paragraph and word is mapped to unique vector. Both paragraph and word vectors are averaged to predict the next words vector. The only difference between this setup and word2vec model is that now there is additional paragraph vector in parallel to word vectors. Paragraph vector D from figure 3.3 acts as memory that tries to learn topic of the context. Words are sampled from paragraph using window size and paragraph vector is shared accross all words. However, the word vectors is shared accross all paragraphs. At each training step, different words using window size are sampled and trained using gradient descent. After training paragraph vectors can be utilized as features to machine learning models. Paragprah vector has very large advantage when the focus is to extract semantic relationships between paragraphs.

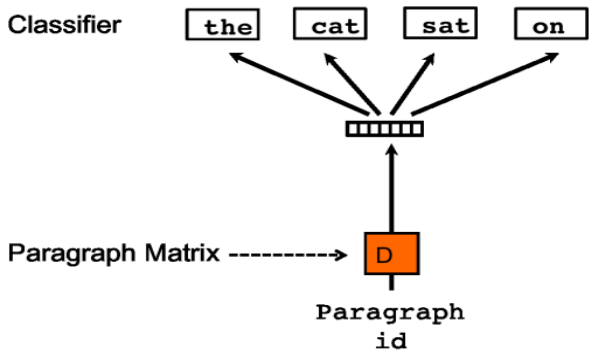
3.2.4.2 Paragraph Vector: Distributed bag of words model

Above method uses sampled context words and document vector to predict next future word. Another method is to keep only document vector and trying to predict sampled words as an output. As can be seen from the figure, the input will contain only document vector and outputs will be sampled tokens.
This model is also similar to Skip-gram word2vec model [3]. Paragraph Vectors outperform bag-of-words models and other text representation algorithms, according to the results from the study [21].

3.3 Classification models

After vectorizing textual data, classification models can be used to classify which type the data belongs to. In order to achieve this classification, several models has been used in this thesis. Two models has been tested, one of them does not take sequentiality into account but another one considers order of words and documents.

Figure 3.3: A framework for learning paragraph vector: Distributed bag of words [21]



3.3.1 XGBoost

A set of models working together to solve a common problem is referred to as ensemble approaches. Ensemble learning, rather than relying on a single model for the optimal solution, makes use of the advantages of numerous diverse methods to compensate for each model's specific weaknesses. The resulting set should be less prone to errors than any single model.

Bagging is an effective ensemble method for reducing variance and, as a result, avoiding overfitting. Ensemble approaches increase model precision by combining many models (or "ensembles") that outperform individual models when employed independently. The Random Forest Classifier is an ensemble technique in which parallel decision trees are used to determine the final categorization type.

A boosting (sequential) ensemble is another strategy, in which the individual decision trees are instead trained consecutively. One of the boosting algorithms is Gradient Boosting where weights are updated using gradient descent. In this method each decision tree model is originally regarded a weak learner, but as a result of training, weak learners become strong. The weak learners here are the individual decision trees. Each tree is connected in a sequence, with each tree attempting to reduce the error of the previous tree. Initially, the sequence's initial model is trained on a random subset of the data's samples. When an observation is incorrectly classified, its weight is updated, while the weights of those that are correctly classified are reduced. Because the likelihood of selecting a mistakenly categorized observation increases, only those observations that were misclassified in model 1 are chosen in the next model.As the loss value decreases, the weights for misclassified observations will be changed, making these weak learners stronger.

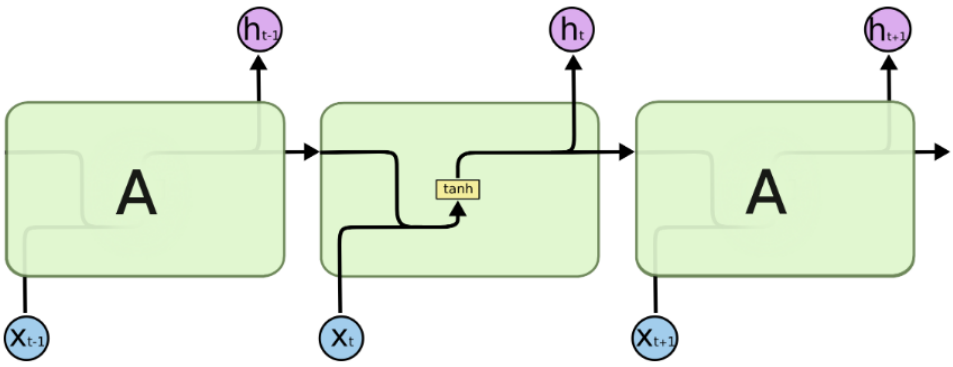
The Extreme Gradient Boosting (XGBoost) [18] machine learning library is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning framework. It is one of the leading machine learning library for regression and classification tasks, and it offers parallel tree boosting. What distinguishes Xgboost from other boosting models are its hardware-level features. During training, tree con-

struction is parallelized using all of CPU cores. The collection of statistics for each column can be parallelized, resulting in a parallel split finding process. Other features include cache awareness, out-of-core computation it results in training faster than other boosting algorithms. To be explicit, XGBoost is a gradient boosting method in and of itself, whereas the underlying model is a sequential tree ensemble model.

3.3.2 LSTM

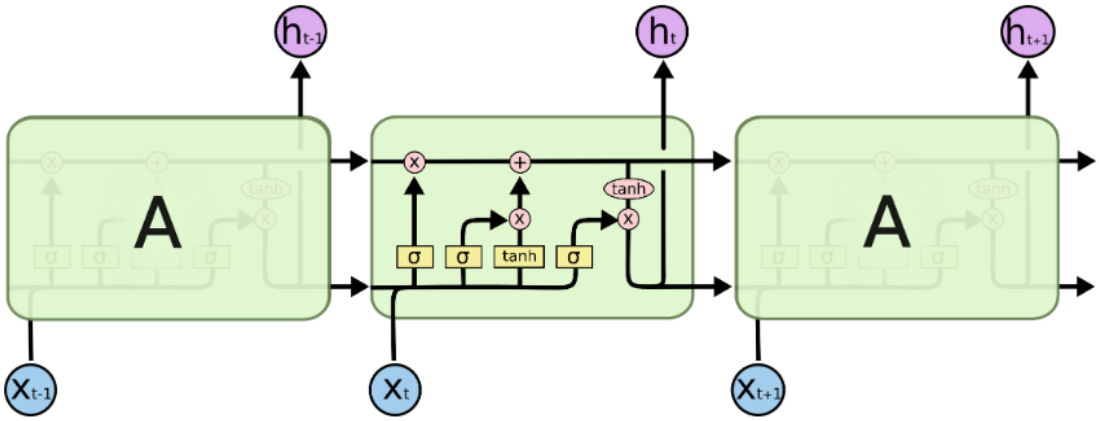
Traditional neural network models does not take short range dependencies of the words into account. When feeding textual data into neural networks, word orders are discarded and model does not learn anything regarding to order of the tokens. This a major shortcomings of traditional neural network models. To migitate this problem Recurrent Neural Network (RNN) have been introduced, which takes sequentiality into account and makes information to persist. Below figure 3.4 shows the unrolled recurrent neural network architecture, each containing single layer with tanh activation function.

Figure 3.4: Unrolled recurrent neural network [19]



RNN is good at learning recent information to perform the present task. If there is a short sequentiality between the inputs then RNN will be able to persist the information through the network from first to last input. However, vanishing gradients are a concern for RNN. When RNN has to train on a long sequence, gradients in the backpropogation phase start to fade until it reaches the first layer weights. Therefore, RNN is struggling handling long-term dependencies.
Long Short Term Memory networks, or "LSTMs," are a type of RNN that can learn long-term dependencies. It first was introduced by Hochreiter Schmidhuber (1997) [20], and now widely used in machine learning area. LSTM are designed to construct long-term dependencies which has a chain like structure as in RNN. Instead of having only a single neural network layer in each chain, there are 4 layers interacting with each other.
The main idea behind LSTM is that it has the ability to remove or add information to cell state, which regulated by structure called gates. Gates helps to persist or prevent the information going to cell states. There are in total 3 gates: forget, input and output gates.

Figure 3.5: Unrolled long-short term memory network [19]



Forget gate uses sigmoid layer to prevent unnecessary information to go through the cell state. It takes previous hidden cell h_{t-1}, current input x_t and outputs a number between 0 and 1 for each coefficient of cell state C_{t-1}. As a result, these cell state values are masked using forget gate and only numbers that are multiplied by 1 goes through the state.

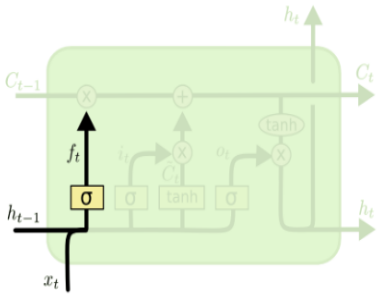
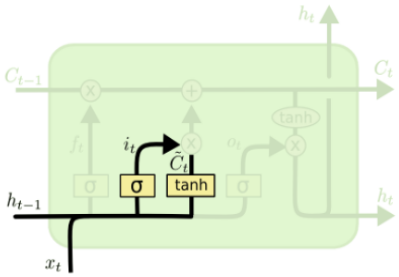


Figure 3.6: Forget gate structure

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.6}$$

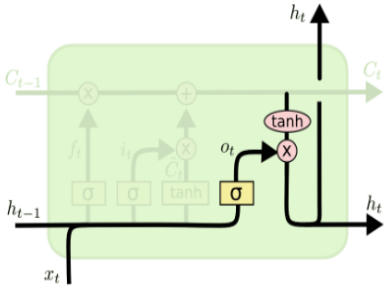
The next step is to decide what new information are going to be stored in cell state. As a first step, sigmoid layer decides which numbers will be updated then tanh layer creates new candidate vectors $\tilde{C}_t$ which will be added to cell state. The old state C_{t-1} should be updated as well to the new cell state. Old state is multiplied by f_t and then summed by $i_t * \tilde{C}_t$.



$$
\begin{aligned}
i_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t
\end{aligned}
\tag{3.7}
$$

Figure 3.7: Input gate structure

The final step is to choose which data to output. The first stage is to decide which cell state numbers should be output using a sigmoid layer. The cell state will then be multiplied by the output of the sigmoid gate via tanh (to force the values to be between 1 and 1), so that we only output the parts we decided to



$$
\begin{aligned}
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{3.8}
$$

Figure 3.8: Output gate structure

Because the problem that needs to be solved has long range dependencies in itself, LSTM networks will be used in this thesis.

3.4 Evaluation

Evaluating the models is part of machine learning pipeline. Evaluation metrics give the idea of how well model performs on given dataset. In this thesis project, the performance of the models will be evaluated by *F1 score* metric. *F1 score* is calculated using another two metrics *Precision* and *Recall*.

Precision is proportion of number of correct positive results to the number of positive results predicted by the model.In another word, its ratio of true positives divided by number of all positive results predicted by classifier.

$$
Precision = \frac{TP}{TP + FP} \tag{3.9}
$$

Recall is the number of correct positive results divided by the number of ground truth positives.

$$Recall = \frac{TP}{TP + FN} \tag{3.10}$$

Finally *F1 score* is a metric that tries to find the balance between precision and recall using following formula:

$$F1\ score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{3.11}$$

Each class will have its own f1 score when evaluating the model. In this thesis **macro F1 score** will be used as an evaluation metric which is average of all classes' F1 scores.

3.4.1 Interpretability

Obtaining a sufficient score is not enough for the model to be approved in the industry. The ability to interpret and explain why the model made certain judgments is essential.
As mentioned earlier, in this thesis two models, *XGBoost* and *LSTM* will be used as a classifier. Various tools will be used to interpret both of the models.

3.4.1.1 Shapley Additive Explanations (SHAP)

Shapley Additive exPlanations (SHAP) is one of those metric that helps to interpret the model decisions. They are based on "Shapley values" created in game theory by Shapley (1953) [24]. When two or more players or components are involved in a strategy to obtain a desired outcome or payoff, it is called game theory. The Shapley value is a solution notion in game theory that equitably distributes both profits and costs to multiple individuals operating in coalition. After exploring every possible combination of movements that one player can make, the Shapley value is the average estimated marginal contribution of one player, where each player may have contributed more or less than the others, Shapley value can help decide a payoff for all of them. SHAP was created expressly for evaluating machine learning model predictions [25]. These are based on the Shapley Values, which are optimal from a game-theoretical standpoint but incorporate a number of efficient estimation methods to make the calculations practical. The Shapley value is calculated at a high level by carefully perturbing input characteristics and seeing how adjustments in the input variables correlate to the model prediction. The average marginal contribution to the total model score is then determined as the Shapley value of a specific feature.

Shap values will be utilized to show the model's feature importance in this thesis. We will have a better understanding of the model decisions and features if we

use shap plots. Shap plots represent the contribution of each feature to the model prediction. Subject Matter Experts must also be aware of the most critical features in the data that models consider important. These features are used to keep their rule-based model up to date.

3.4.1.2 t-distributed Stochastic Neighbor Embedding (t-SNE)

To classify the target values, machine learning models often use high-dimensional features. When feature space is high-dimensional, however, it becomes difficult to grasp the relationships between features. Word2vec embeddings, for example, generates an N-dimensional vector for each word. Dimensionality reduction should be used to lower the dimension from N to 2 in order to visualize these vectors and see which words are close to each other. We can observe which words are close to each other and how they might be clustered in a 2D plot by using dimensionality reduction. This visualization will provide us with further information regarding feature vectors.

The paragraph vectors must be interpreted in some way because doc2vec will be utilized in this thesis. Doc2vec outputs a high-dimensional vector space which is not easy to visualize in order to understand the relationships between semantics. t-Distributed Stochastic Neighbor Embedding [22] is a dimensionality reduction method that used in this thesis to reduce the high-dimensional feature vectors to low-dimension such as 2 or 3D. The doc2vec embeddings derived from data will be better understood as a result.
There are 3 steps in t-SNE algorithm in order to reduce the high dimensional data to low dimensional space.

- The first step is for each point to measure how likely other points are neighbors. Meaning that for euclidean distance will be calculated from each point to all other points. Then, these distances will be fed into conditional probabilities to measure the similarities between every two point. Conditional probability using gaussian distribution will be calculated as follows:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{3.12}$$

  After finding the conditional probabilities, joint probability should be measured using the following formula:

$$p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{3.13}$$

- The second step is to project points to low-dimension and calculate joint probability distribution. First, points as the number of data points will be initialized with K features randomly where K is the dimension of target space. Again joint probability will be calculated for each two point but now using t-distribution

$$q_{j|i} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||y_i - y_j||^2\right)^{-1}} \tag{3.14}$$

- The last stage will be to make the low-dimensional joint probability distribution of the data points as close to the original data points as possible. To achieve this, Kullback-Leiber divergence will be used as mentioned in paper [22]. Kullback-Leiber divergence measures how dissimilar two distributions are. KL divergence for P and Q distributions is measured as follows:

$$D_{KL}(P||Q) = \sum_{x \epsilon X} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{3.15}$$

  At last, gradient descent is used to minimize KL divergence of the joint probability distribution P (high-dimensional) and Q (low-dimensional space. By doing this optimization, we achieve dimensionality reduction from high-dimensional space to low-dimensional space.

$$C = KL(P||Q) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}} \tag{3.16}$$

3.4.2 K-Fold Cross Validation

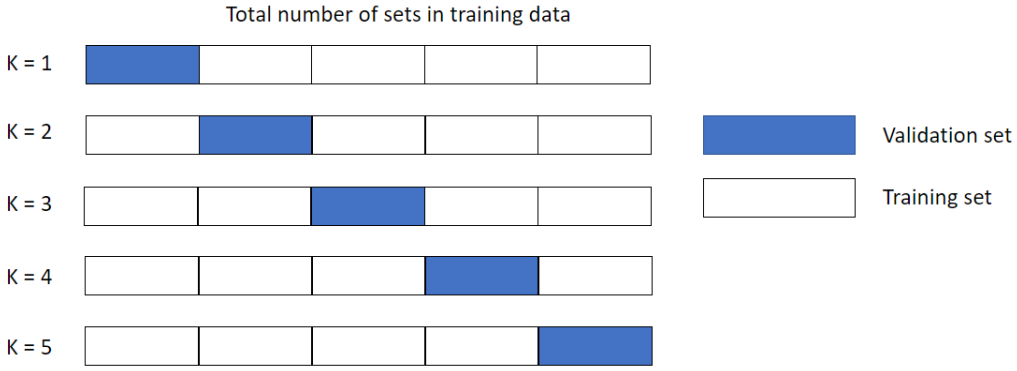
Fitting the ML models to the data isn't enough. Validating the performance on a different set that the model has never seen before is critical. This allows one to determine whether or not a machine learning model generalizes well. Therefore, the data should be divided into training and test set.

This ensures that the model does not overfit on the training data and that it generalizes well on the test data. If we simply divide the data into training and test sets, we may strive to maximize the model's performance on the test set by picking the appropriate parameters. The problem is that we risk overfitting on test data, which means we focus on how to increase performance based on a specific test set. If we have a completely new, previously unknown set, the model may perform poorly on it. Validation set with K-fold cross validation technique might be utilized to overcome this difficulty and generalize better. Another key benefit of cross validation is that it allows us to determine the best hyperparameters for the classifier model. We may overfit the test data by attempting to maximize the score on a certain test set if we try to find hypterparameters without applying cross validation. Cross validation, on the other hand, aids in greater generalization and the discovery of the best parameter combination for the model.

In K-Fold cross validation, data is divided into k folds. Each time one of the k subsets is used a validation set and other k - 1 sets form the training set. Each time model trains on training set and validates on validation set. The model will be trained k times in total and average of the k F1-scores will be used as a metric. This considerably decreases bias because the majority of the data is used for fitting, and it also significantly reduces variance because the majority of the data is also

included in the validation set. Below figure shows how data can be splitted into 5 folds using K-fold cross validation technique:

Figure 3.9: K-fold cross validation where K = 5



The information used in this thesis is unbalanced. As a result, having the same proportion of classes in the training and test sets is critical. It should be noted that the Ericsson data contains duplicate serial numbers. As previously stated, these serial numbers represent a certain radio unit. The log contents of these duplicate serial numbers may overlap. As a result, it's crucial to avoid putting data rows with the same serial numbers into different sets. The reason for this is that if we distribute duplicate serial numbers to the training and test sets, the test set will not be completely unseen data because it will include some overlapping parts from the training data. To overcome this problem, same serial numbers should be put into the same sets. We can achieve this using GroupShuffleSplit provided by scikit-learn framework [23] which will split data based on serial numbers.

4

Methods

The most crucial part of data preparation is dividing and labeling the data into intervals. As previously stated, each unit contains multiple logs, each of which contains multiple lines with appropriate timestamps. The log lines will be divided into distinct intervals and labeled with the appropriate alarm label using timestamps from log lines and alarm types. The following table is used as a fragment from the original data, with the later sections demonstrating how this snippet will be divided and labeled. For a certain radio unit at the customer network, the table shows log lines and triggered alert categories.

4.1 Data preparation

The initial step in the ML pipeline is to prepare and preprocess the data before feeding it to the machine learning model. As mentioned before the data used in this thesis contains a set of units with appropriate labels, each of which has several log files expressed as text files containing the log lines. Each unit can have one or more labels, each with its own timestamp indicating when the alarm was triggered. The following section will describe the various data preparation approaches that have been used throughout the thesis.

4.1.1 Divide and Label

The most crucial part of data preparation is dividing and labeling the data into chunks. As previously stated, each unit contains multiple logs, each of which contains multiple lines with appropriate timestamps. The log lines will be divided into distinct intervals and labeled with the appropriate alarm label using timestamps from log lines and alarm types. The following table is used as a fragment from the original data, with the later sections demonstrating how this snippet will be divided and labeled. For a certain radio unit at the customer network, the table 4.1 shows log lines and triggered alert categories.

Table 4.1: Snippet row from data

Serial	Log lines	Timestamps	Alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4> [210503 204135] <log message 5> [210504 020511] <log message 6> [210504 024345] <log message 7> [210506 054005] <log message 8>	2021-05-02 06:49:56 2021-05-04 03:08:12	DU raised alarm Radio fault

Because the thesis' principal goal is to forecast an alarm before it occurs, fault alarm times will be altered to meet this goal. For example, if the alarm occurs at T time and we want to anticipate it one day ahead of time, the alarm time will be pushed 1 day backwards, and only log lines with timestamps smaller than T - *1 day* will be used as data. The table 4.2 represents how the alarm times change when shifted time is *1 day*. When data is divided and labeled based on 1 day shifted time, the last four log messages will be omitted.

Table 4.2: Snippet row from data with shifted timestamps

Serial	Log lines	Timestamps	Alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4> [210503 204135] <log message 5> [210504 020511] <log message 6> [210504 024345] <log message 7> [210506 054005] <log message 8>	2021-05-01 06:49:56 2021-05-03 03:08:12	DU raised alarm Radio fault

Expanding window

Expanding the window is method used to create a forecast model by generating a statistic from all available historical data. It's an expanding window because it becomes bigger as more data is gathered. In this thesis, this approach has been used to divide the log lines into intervals. Each alarm type in the data will be assigned its own chunk. As can be seen from table 4.1, each log lines and alarm types has its own timestamp information which can be used to divide the log lines into bags. The following figure will show overview of the expanding window approach that has been applied on the Ericsson data.
Each alert type is divided into chunks, as shown in the figure above. The history data and additional log lines will be used as a second chunk for the second alert since this is an expanding window technique. Because there are no more alarms following the second alert, the log lines after that will be omitted. The table below shows how the original data for a specific unit was altered using the expanding window method.

Figure 4.1: Overview of the expanding window approach

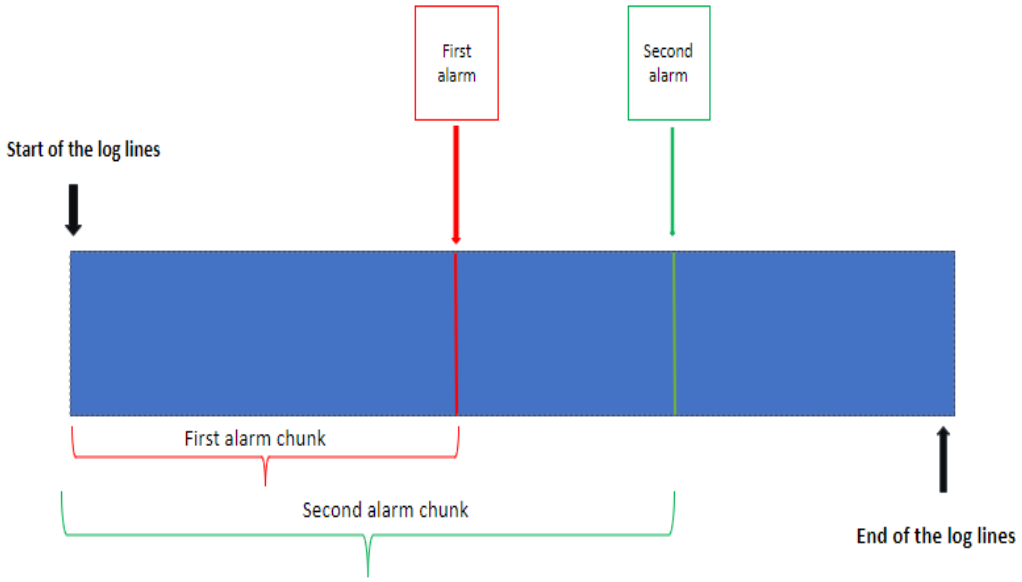


Table 4.3: Data fragment after expanding window approach

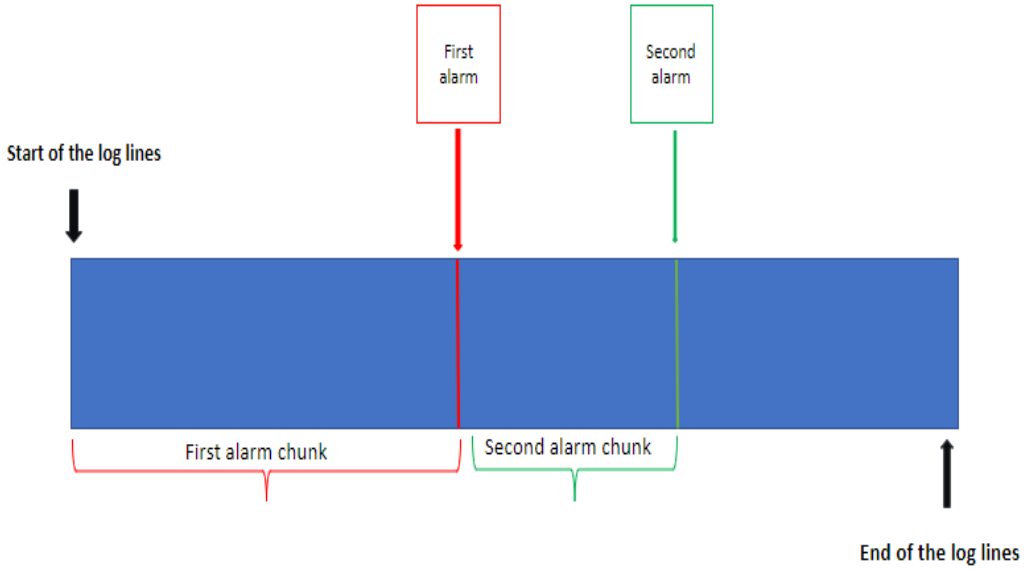
Serial	Log lines	Alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4>	DU raised alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4> [210503 204135] <log message 5> [210504 020511] <log message 6> [210504 024345] <log message 7>	Radio fault

As can be seen in the table 4.3, the expanding window strategy yielded two rows. Based on timestamp comparisons, each row reflects the alarm and log lines that were retrieved from the original log lines 4.1. Since this is expanding window approach, the log lines from the first chunk will also be included in the second chunk.

Independent chunks

In this strategy, no preceding chunk's history data will be used for the following chunk. This means that each chunk will have its own log lines, separate from the other chunks. The following figure will show overview of the independent chunks approach that has been applied on the Ericsson data.
Each alert type is divided into separate chunks, as shown in the figure above. The log lines after the second alert will be eliminated because there are no more alerts.

Figure 4.2: Overview of the independent chunks approach



The following table shows how original data for a particular unit has been changed using expanding window approach.

Table 4.4: Data fragment after independent chunks approach

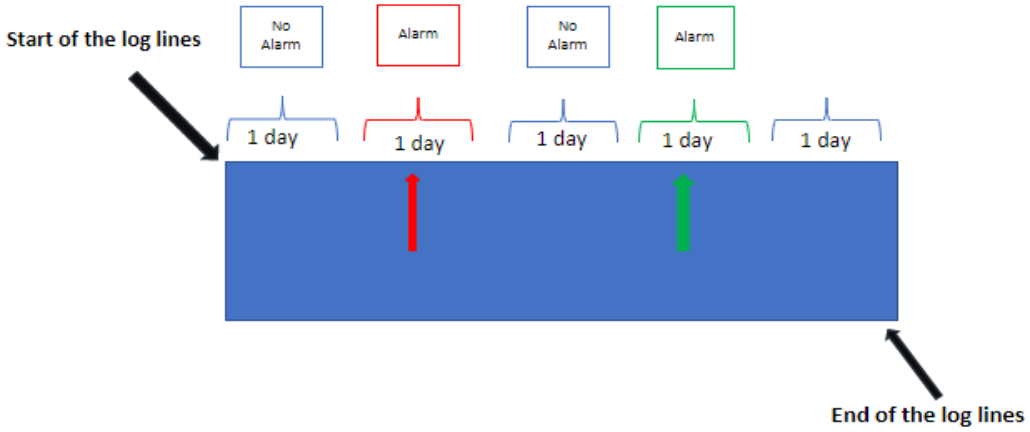
Serial	Log lines	Alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4>	DU raised alarms
E552852522	[210503 204135] <log message 5> [210504 020511] <log message 6> [210504 024345] <log message 7>	Radio fault

As can be seen in the table 4.4, the independent chunks strategy yielded two rows. Each row indicates the alarm and log lines that were extracted from the original log lines based on timestamp comparisons. The log lines from the first chunk will not be included in the second chunk since this is an independent chunks technique where each chunk should be independent of each other. Because it is generated after the last alarm, the last log line with the content *<log message 8>* has been omitted.

Chunks with Time intervals

Last approach is that the logs can be divided into time interval chunks, with each time interval defined as 5 minutes, 6 hours, 1 day, 2 days, and so on. The following figure represents the overview of this approach with 1 day intervals.
After the last alarm, 1-hour chunks will be omitted and not be part of the ml pipeline. In this method, each chunk will have its own log lines that occurred during that time

Figure 4.3: Overview of the time interval chunks approach



frame. The table below shows how the original data fragment changes as a result of this method.

Table 4.5: Data fragment after independent chunks approach with time interval

Serial	1st chunk	2nd chunk	3rd chunk	4th chunk	Alarms
E552852522	[210501 123516] <log message 1> [210501 123516] <log message 2> [210501 172613] <log message 3> [210501 203936] <log message 4>		[210503 204135] <log message 5>	[210504 020511] <log message 6> [210504 024345] <log message 7>	No Alarm DU raised alarms No Alarm Radio Fault

As can be seen from the table 4.1, the original data fragment can be divided into 6 days, with the first date being *2021-05-06* and the latest date being *2021-05-06*. However, there are only four chunks in the above table, each representing a different day. The absence of two chunks is due to the fact that the last alert occurs on *2021-05-04* and those two days after the last alarm are omitted.

4.1.2 Data Parsing

In this section, various parsing algorithms will be applied to a data fragment to demonstrate how parsing alters the data structure. Based on the parsing procedures, unstructured text lines will be turned into structured form. The table below represents a snippet of the data that will be converted to a structured form using various parsing algorithms in subsequent subsections.

Table 4.6: Data fragment from unstructured log lines

Unstructured log lines
[200811 051847] 1: RU start/restarted; Restart cause:COLD; LMC ID: CXP901096%14_R63DL14
[200811 051847] 1: RU start/restarted; Restart cause:ORDERED; LMC ID: CXP9017316%2_R66GC
[200811 051847] 8: COLI command: trdc setup3 1 0 1 2112500
[200811 051847] 8: COLI command: fm collect
[200811 051847] 8: COLI command: trdc on 1

Regex Parsing

Regex parsing has been used to filter out dynamic tokens such as timestamps, ip addresses, numbers and etc by using manual regex patterns. The table below shows how the original data fragment was transformed from unstructured to structured using regex parsing.

Table 4.7: Processed log lines using Regex parsing

Regex parsed log lines
RU start/restarted Restart cause COLD LMC ID CXP901096%14_R63DL14
RU start/restarted Restart cause ORDERED LMC ID CXP9017316%2_R66GC
COLI command trdc setup3
COLI command fm collect
COLI command trdc on

Advanced Parsing

To recognize dynamic tokens automatically, advanced parsing algorithms such as Awsom-LP and Drain will be used. These parsing algorithms have a higher temporal complexity than simple regex parsing, but they provide a more structured format for the log lines. The table below shows how the original log lines from the table alter when advanced parsing is used.

Table 4.8: Structured log lines with Advanced parsing

timestamp	Advanced parsed log lines	parameters
200811 051847	RU start/restarted<*> Restart <*> LMC ID<*> <*>	Cause, Cold, CXP901096%14_R63DL14
200811 051847	RU start/restarted<*> Restart <*> LMC ID<*> <*>	Cause, Ordered, CXP9017316%2_R66GC
200811 051847	COLI command<*> trdc setup3 <*>	
200811 051847	COLI command<*> fm <*>	collect
200811 051847	COLI command<*> trdc <*> <*>	on

4.2 Feature Extraction

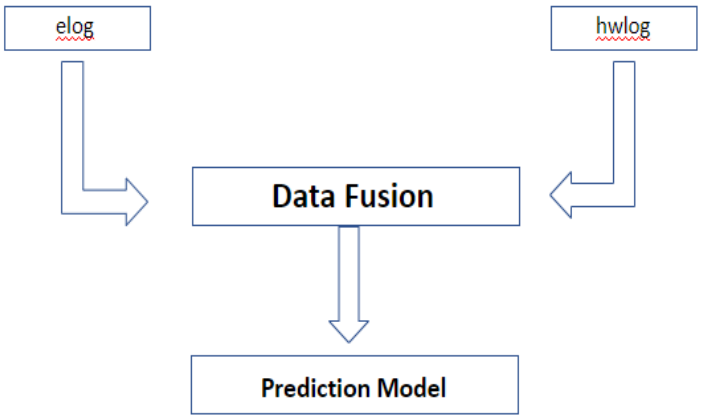
The ML pipeline's next step is to extract features from parsed log lines, which involves converting log lines into numerical representations. This was accomplished using Bag-of-Words, TF-IDF, and Doc2Vec embeddings. Both Bag-of-Words and TF-IDF can be computed using **CountVectorizer** from the Python library Scikit-learn [23]. CountVectorizer generates its vocabulary for each log type separately. On the other hand, doc2vec from **gensim library** [26] has been used to extract vector embeddings from each log type's documents. The next paragraphs outline some key parameters for both feature extraction approaches.

4.3 Model selection

Finding solutions to handle multimodality data was critical given the data has two log types. It should be pointed out that each hwlog file has an average of twenty-one lines, but each elog file has three thousand. There are two apporoaches to handle multimodality of different log types: *Early fusion* and *Late fusion*. The late fusion model has the disadvantage of not being able to use the relationship of features between log types in its prediction. Instead, it creates a prediction for each of the log types, then adds the probabilities together to make the final prediction. If some log types don't have valuable information for certain logs, their prediction will simply confuse the model rather than help it. In comparison to elog, hwlog has less lines in the dataset, and there are even some missing hwlog files.

In this thesis, an early fusion strategy is employed to fuse multiple data sets before predicting the alarm type. Before training the model, the several log type features (elog and hwlog) will be concatenated into a single vector. The diagram 4.4 depicts the structure of the early fusion technique employed in this thesis.

Figure 4.4: Early fusion approach

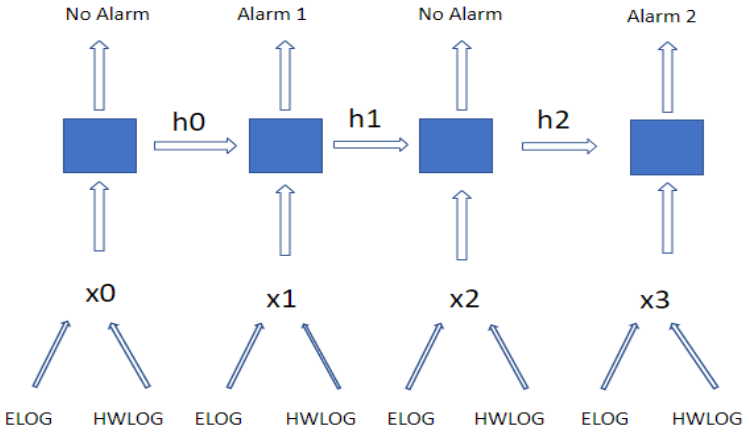


The baseline solution is an initial approach used in this thesis to provide a baseline result for the later approaches. The gradient boosting model XGBoost was chosen as the baseline model. Another model employed in this thesis is the XGBoost model but now with independent chunks method with no dependency between chunks. Each chunk represents a separate data point that will be fed into the Xgboost model to predict the type of alarm.

Long-short term memory networks on time interval chunks was the final approach used in this thesis. Many-to-many lstm model has been built where each row containing several chunks and alarms fed into this network sequentially. Doc2vec embeddings are used to extract vectors from each log type before predicting label for each chunk. As can be seen from the figure 4.5, the elog and hwlog doc2vec vector embeddings for each chunk are concatenated, then fed into the lstm layer, and lastly the softmax layer is used to identify the chunk type. The hidden layers aid

in maintaining chunk dependancy; for example, the weights obtained from the first lstm layer contribute to the training of the next, and so on. Log lines are divided into one-day chunks, as previously stated. For each data row, an LSTM network requires a specified length for the number of chunks that will be trained. After dividing and labeling each row log line, we now know how many chunks were produced for each row based on one-day intervals. The maximum number of chunks (max len) is utilized as the LSTM network's defined fixed length. Padding and masking techniques are used to pad the rows whose number of chunks are less than max_len. Because the LSTM model demands that all data points have the same shape, they should be padded such that all rows have the same dimension. During training, those padded vectors are masked so that they do not contribute to the gradient during the backpropagation step.

Figure 4.5: Overview of the many-to-many lstm architecture for the example made in Table 4.5



4.4 Workflow

As mentioned in previous sections there are multiple approaches in this thesis, and results from these approaches are demonstrated in the next section. The workflow for ml pipeline is as follows:

1. Baseline Xgboost model
    (a) Divide and label log lines using *Expanding window* approach
    (b) Parse logs using either Regex, Drain or Awsom-LP parsing methods
    (c) Extract feature vectors using BOW or TF-IDF
    (d) Fit Xgboost model
    (e) Evaluate result using F1-score metric

2. Xgboost model with Independent chunks

   (a) Divide and label log lines using *Independent chunks* approach
   (b) Parse logs using either Regex, Drain or Awsom-LP parsing methods
   (c) Extract feature vectors using BOW or TF-IDF
   (d) Fit Early-fusion Xgboost model
   (e) Evaluate result using F1-score metric

3. Experimental LSTM model

   (a) Divide and label log lines using *Time interval* chunks approach
   (b) Parse logs using either Regex, Drain or Awsom-LP parsing methods
   (c) Extract feature vectors using Doc2Vec embeddings
   (d) Fit LSTM many-to-many model
   (e) Evaluate result using F1-score metric

5

Results

The baseline and experimental model results were reported in this section. The interpretability of these models provided for a better understanding of the model's core features.

5.1 Baseline Xgboost model

The baseline model with expanding window approach has been trained and the validation results are presented in Table 5.1. There are ten shifted time variations that have been employed with log parsers and word embeddings. The initial shifted time is zero because it reflects predicting the type of alarm (*raised alarms* or *radio faults*) exactly when it happens. The model was then trained for a period of one, two, three days, and so on, in order to predict failure before it occurs. The shifted time indicates how many minutes, hours, or days the alarm timestamps have been pushed back.

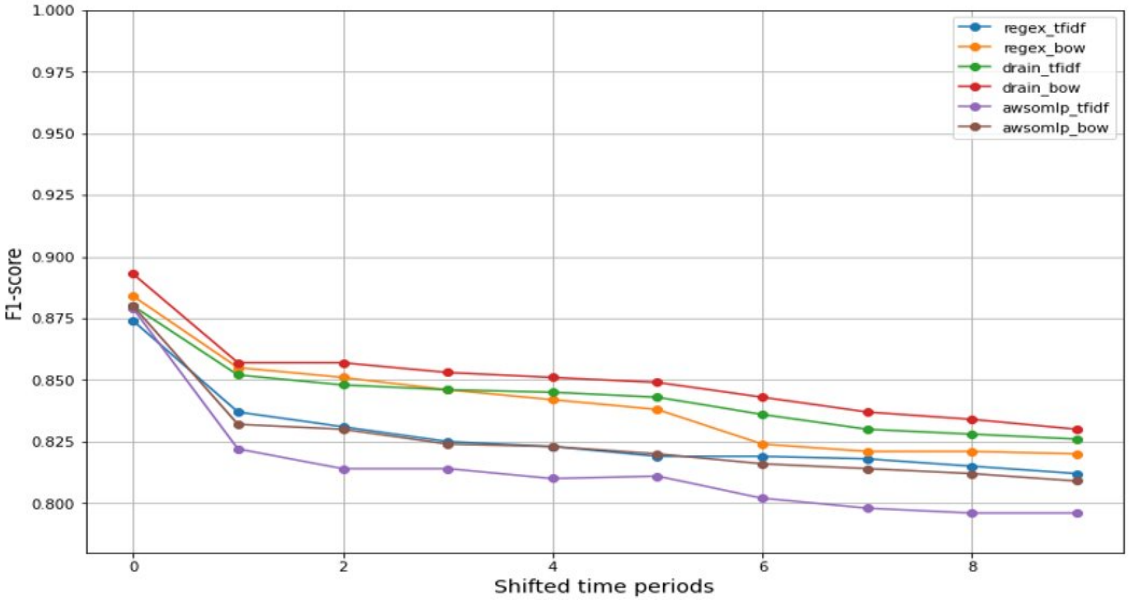
Different log parsers such as Regex, Drain and Awsomlp has been used to parse the logs before using word embeddings. Drain and Awsomlp parser methods helped to detect dynamic tokens and remove them from log lines. As word embeddings, TFIDF and BOW have been employed. Only onegrams were used to generate the vocabulary for both techniques. The parameter *max_features* has been set to 1000, indicating that the vocabulary will be limited to the top 1000 frequently used words. F1-score evaluation metric has been used and all the results are average of F1-scores for two categories, *raised alarms* and *radio faults*.

As can be seen from the Table 5.1 and Figure 5.1, Drain log parser with BOW embeddings dominates the table with **0.854** average f1-score. Except when the shifted time is 1 day, Drain with BoW embedding outperforms any other combination of parser and embeddings across the shifted times. When the shifted time is 1 day, the Regex log parser with BOW embedding outperforms other combinations. As indicated by the average results across shifting time, log parsers with BOW embeddings beat log parsers with TFIDF embeddings. When using the Xgboost model with expanding window technique, we can conclude that BOW embeddings perform better than TFIDF for this dataset.

Table 5.1: Xgboost model results using expanding window approach. 5-fold cross validation has been used

Shifted time \ Combinations	Regex TFIDF	Regex BOW	Drain TFIDF	Drain BOW	Awsomlp TFIDF	Awsomlp BOW
0 days	0.874	0.884	0.880	**0.893**	0.879	0.880
1 day	0.837	0.855	0.852	**0.857**	0.822	0.832
2 days	0.831	0.851	0.848	**0.857**	0.814	0.830
3 days	0.825	0.846	0.846	**0.853**	0.814	0.824
4 days	0.823	0.842	0.845	**0.851**	0.810	0.823
5 days	0.819	0.838	0.843	**0.849**	0.811	0.820
6 days	0.819	0.824	0.836	**0.843**	0.802	0.816
7 days	0.818	0.821	0.830	**0.837**	0.798	0.814
8 days	0.815	0.821	0.828	**0.834**	0.796	0.812
9 days	0.812	0.820	0.826	**0.830**	0.796	0.809
Average	0.827	0.840	0.843	**0.850**	0.814	0.826

Figure 5.1: Combination of log parsers and word embeddings using Expanding window approach with Xgboost model



5.2 Xgboost model with Independent chunks

Table 5.2 represents the early fusion xgboost model trained using independent chunks approach on different shifted time periods. Log parsers such as Drain, Awsomlp and Regex again has been used with word embeddings such as TFIDF and BOW. The same parameters for word embeddings has been used as in the previous setup. F1-score evaluation metric has been used and all the results are average of F1-scores for two categories, *raised alarms* and *radio faults*.

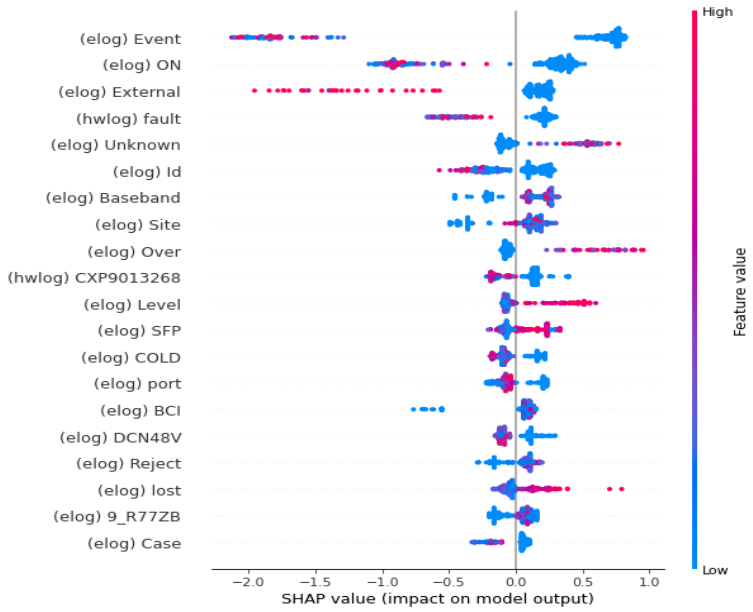
As can be seen from Table 5.2 and Figure 5.3, across the shifted time, two approaches are ruling the table. In all shifted time and average results, TFIDF and

BOW embeddings with Drain log parser surpass other combinations. We can see from Table 5.2 that there are no significant differences between BOW and TFIDF embeddings when we look at each parser average outcomes across shifting time. If we look at the average results across shifted time periods, Drain log parser with BOW embedding is the best approach with *0.873* average F1-score. We can see big drops from shifted time 0 to 1 day for each combination. The reason is that when the time is pushed 1 day back, we lose very vital indications towards the alarm. Because when an alarm goes off, there are usually some extremely important indicators. 5, 10, or 30 minutes prior to that suggests that there is a problem or that something will go wrong very soon. Because the log lines following shifted time are not used, such important features are missed which results in a significant decline in the f1-score between 0 and 1 day shifted time.

Table 5.2: Xgboost model results using Independent chunks approach. 5-fold cross validation has been used

Combinations / Shifted time	Regex TFIDF	Regex BOW	Drain TFIDF	Drain BOW	Awsomlp TFIDF	Awsomlp BOW
0 days	0.933	**0.938**	0.932	0.934	0.923	0.928
1 day	0.841	0.847	**0.874**	0.865	0.832	0.834
2 days	0.836	0.839	**0.865**	0.862	0.830	0.833
3 days	0.833	0.836	**0.863**	0.860	0.827	0.831
4 days	0.831	0.830	**0.860**	0.860	0.826	0.829
5 days	0.830	0.827	0.857	**0.859**	0.823	0.824
6 days	0.827	0.824	0.855	**0.857**	0.822	0.824
7 days	0.827	0.822	0.854	**0.856**	0.821	0.820
8 days	0.824	0.820	0.848	**0.851**	0.820	0.820
9 days	0.817	0.818	0.845	**0.848**	0.818	0.819
Average	0.838	0.840	0.865	**0.866**	0.834	0.836

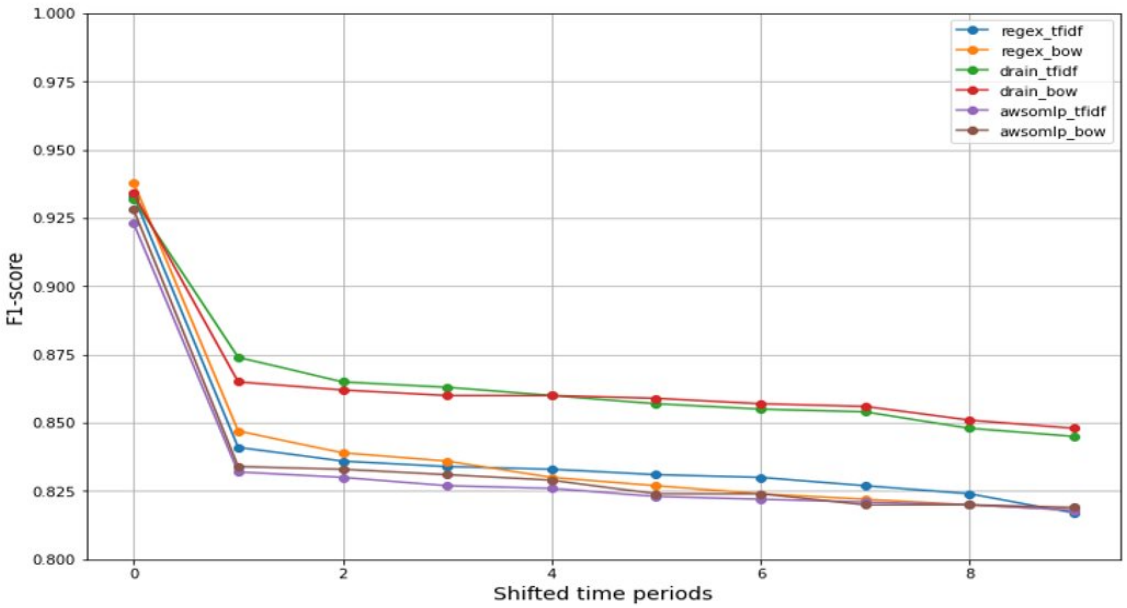
Figure 5.2: Shap summary plot for Xgboost model



We can see from the Figure 5.2 the shap summary plot for the Xgboost model used with independent chunks approach. Top 20 most important features on the left side

are presented in descending order. The feature values are displayed in a color bar on the right, ranging from high to low. There are occurrence values for each feature in BOW embeddings, for example. If a feature does not occur frequently in a row, that row will be represented by the color blue; otherwise, it will be represented by the color red.
In front of each feature, we can observe the feature type *elog* or *hwlog* in parenthesis. On the x-axis there is shap value impact showing to which class does the feature contributes more. If we look at the first feature which is *Event* from elog log lines, we can see that low values for this feature impacts on model towards the right class which is *raised alarms*. However, high values contributes to the left class which is *radio faults*. The elog features are dominating the table since 18 out of top 20 features are extracted from elog lines. We can see feature named as *fault* in top 5 features which was extracted from hwlog. High values for *fault* contributes to the *radio fault* class.

Figure 5.3: Combination of log parsers and word embeddings using Independent chunks approach with Xgboost model



5.3 Experimental LSTM model

Table 5.3 represents the lstm model trained using time interval chunks approach on different shifted time periods. As mentioned in section 4.1.1, the 1 day interval has been used to divide the logs into 1 day chunks. Instead of word embeddings like BOW and TFIDF, doc2vec paragraph vector embedding has been used to extract features from log lines.
F1-score evaluation metric has been used to track the model progress based on training and validation results. *Categorical crossentropy loss* has been used since the focus is multi-class classification of *raised alarms* and *radio faults*.

Table 5.3: LSTM model with 1 day interval chunks approach

Combinations / Shifted time	PV-DM	PV-BOW
0 days	0.878	**0.889**
1 day	0.847	**0.862**
2 days	0.846	**0.860**
3 days	0.845	**0.858**
4 days	0.842	**0.856**
5 days	0.831	**0.850**
6 days	0.824	**0.846**
7 days	0.821	**0.844**
8 days	0.816	**0.838**
9 days	0.795	**0.832**
Average	0.834	**0.853**

Two doc2vec embeddings, *distributed memory* and *bag of words*, has been used to generate vectors from documents. As can be seen from Figure 4.5, both elog and hwlog vectors are concatenated for each 1 day chunk interval, and then an LSTM model with 32 neurons is added on top of each concatenated input. The L2 kernel regularizer with parameter *0.0001* is used to reduce overfitting. With a learning rate of *0.001*, gradient descent with Adam optimizer is utilized during model training. We can see from Table 5.3 that except when the shifted time is 3 days, *paragrapch vector with bag of words* approach is dominating the table in almost all shifted time periods with average **0.853** F-1 score.

5.4 Comparison

As can be observed from the previous sections, three main approaches, expanding window, independent chunks and chunks with time interval, has been tried during the thesis. It should be noted that the first two approaches are compared using a dividing and labeling procedure rather than models. For both approaches, the same Xgboost model was used, but alternative dividing and labeling techniques were used. As a result, comparing the last approach to the first techniques isn't fair because the last strategy uses a different dividing and labeling technique and model than the first two. It would be preferred to train all models on all possible splitting approaches, but due to time constraints, this was not achievable during the thesis period.

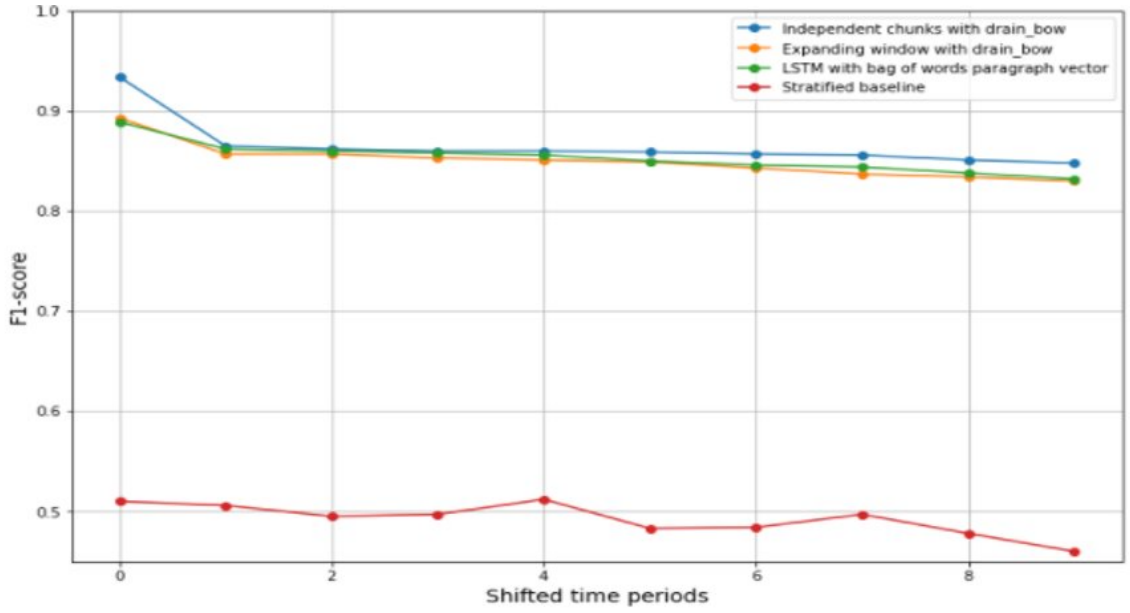
The best result obtained from each strategy is compared in this section to determine which approach works best on average. Drain log parser with BOW embedding dominated the table both in expanding window and independent chunks approach. When employing the LSTM model, a graph vector with the BOW model performs better than a distributed memory technique. These three techniques are compared, as well as a stratified dummy classifier, to see if we can do better.

Table 5.4: Comparison of best approaches obtained from three strategy

Combinations / Shifted time	Xgboost with Expanding window Drain BOW	Xgboost with Independent chunks Drain BOW	LSTM PV-BOW	Stratified baseline
0 days	0.893	**0.934**	0.889	0.510
1 day	0.857	**0.865**	0.862	0.506
2 days	0.857	**0.862**	0.860	0.495
3 days	0.853	**0.860**	0.858	0.497
4 days	0.851	**0.860**	0.856	0.512
5 days	0.849	**0.859**	0.850	0.483
6 days	0.843	**0.857**	0.846	0.484
7 days	0.837	**0.856**	0.844	0.497
8 days	0.834	**0.851**	0.838	0.478
9 days	0.830	**0.848**	0.832	0.460
Average	0.850	**0.866**	0.853	0.492

As can be seen from Table 5.4 and Figure 5.4, all the models perform better than stratified dummy classifier. In various shifting time periods, the Xgboost model with independent chunks approach using Drain log parser and BOW embedding surpasses all other models with average 0.873 F1-score across shifting time periods, hence it can be regarded the best approach.

Figure 5.4: Comparison of the best models obtained from each dividing and labeling approach



6
Discussion

As can be seen from Table 5.4, Xgboost model with Independent chunks approach using Drain log parser and BOW embedding is the best result for the dataset **Radio 2219**. Two main divide and label approaches, expanding window and independent chunks, were researched for Xgboost model in this thesis. According to Table 5.4, Xgboost model with Independent chunks approach performs better than Expanding window approach. The early fusion Xgboost model was utilized, which implies that instead of generating separate models for each log type feature, one Xgboost model was developed using the concatenation of the several log type features. It should be pointed out that no manually extracted features by Subject Matter Experts has been used in this thesis. Instead, data driven approaches parsers and word embeddings has been developed to extract vital features from logs. NLP techniques are crucial since they are not confined to any domain expertise and will take into consideration any feature that is important for the model to make decisions. Previously, manually extracted features are used at Ericsson as an input to model and these features are confined to domain knowledge; previously when a new important log line appeared, it was ignored if it did not appear in the list of manually extracted features. But using data-driven approach we are not limited to the list of manually extracted features anymore.

For both divide and label approaches, a variety of log parsers have been tested. We can see from Table 5.1 and Table 5.2 that Drain log parser performs better than other log parsers in both of divide and labeling approach. This means distinguishing between dynamic and static tokens using Dran log parser, as well as excluding dynamic tokens from log contents, helps the model make better decisions. We can also observe that not all automatic parsers outperform a simple Regex parser based on the results. For the **Radio 2219** dataset, the AwsomLp parser with word embeddings performed lower than the simple Regex parser. If we look at Xgboost model performances from Table 5.1 and Table 5.2, we can observe that BOW outperforms TFIDF in practically every combination of log parsers and word embeddings. That is to say, simply calculating the frequency of each feature in a log is sufficient for the model to make better decisions.

Using NLP approaches could lead to a better understanding of the data as well. Model interpretability can be utilized to extract important features that the prediction model deems relevant when making judgments. The main aspects of both BOW and TFIDF word embeddings employed with the Xgboost model is that they are straightforward to interpret and understand. The top 20 features, which were

retrieved from elog and hwlog, are shown in Figure 5.2. **Event**, **On**, **External**, **fault** and other important features let domain specialists better comprehend the logs and even adjust the manual feature extraction list.

The last method employed in this research was dividing logs into one-day intervals and label them as **No alarm**, **Radio faults** or **Raised alarms**. As mentioned before, comparing the last approach to other approaches is not fair since different model and splitting techniques has been used. In the last approach, apart from the dividing and labeling the logs, the key difference was to use different embedding for LSTM model. Document2vector has been used to extract vectors from each log document. In comparison to the XGBoost model, LSTM results in a significantly longer training period since extracting vectors from documents takes more time than BOW embedding. Another limitation of the LSTM model over the XGBoost approach is its interpretability. Since doc2vec is used to extract features, it's difficult to tell which tokens and words influence the model's choice. The reason behind this is that instead of individual word vectors, the LSTM model employs vectors taken from documents.

6.1 Future Work

Only **elog** and **hwlog** were used in the thesis, however there is another log type, **telog**, that can be utilized when extracting futures. Telog is log file type which is constantly produced in the unit, but only collected if a critical alarm is triggered. If there is a memory shortage, this log could be over-written. Telog was not used in this thesis since several of the telog line timestamps occurred after the last log alarms, causing them to be omitted during divide and label. In the future, one could explore and consider ways to incorporate or extract features that are still relevant to this task.

While dividing and labeling the log lines using time interval chunks approach, all log lines before the last 3 month were merged in a 1 chunk. The log lines begin in the year 2016 or 2017, and the quantity of one-day sequences from 2017 to 2022 is a massive amount to train with a CPU, and even with a GPU, the training time is questionable. All log lines that appeared before the last three months have been consolidated into one chunk to speed up the training. In the future, instead of combining log lines before three months, one may explore and attempt various merging time intervals. For example, instead of merging log lines before three months, one could use one month or one year to get a better result. Another important point is that in the last approach of dividing and labeling log lines, logs are divided into one day intervals. In the future, one may investigate alternative time intervals, such as one hour or two days, to determine if they aid in improving model performance.

Due to time constraints, this thesis did not include the use of Xgboost and the LSTM model on all feasible splitting approaches. In the future, one might use the LSTM one-to-one model on the first two techniques to compare the model performance of LSTM and Xgboost fairly.

7
Conclusion

This thesis has demonstrated that data-driven NLP methods combining models like Xgboost and LSTM may be used to forecast important alarms on Ericsson radio units one to nine days before they occur employing logs which created by their hardware and software systems. This was done using different log splitting techniques such as expanding window, independent chunks, log parser methods such as Regex, Drain, Awsom-LP and word embedding methods such as BOW, TFIDF and Doc2Vec to extract important features from logs. To summarize, the Independent chunks splitting technique combined with Drain log parser and BOW word embedding utilizing the Xgboost model outperformed all other approaches. Furthermore, while this BOW word-count format is simple, it yields interpretable model predictions when evaluated using SHAP values. These words are crucial for Subject Matter Experts since they provide insightful information regarding model predictions. During this thesis period, NLP data-driven techniques were applied, which required less domain expertise knowledge and reduced domain export effort while developing a model pipeline.

It's worth noting that using doc2vec embeddings with complex models like LSTM yielded promising results. This indicates that there are semantic linkages between log chunks, and the doc2vec embedding was able to extract the document embeddings utilized by LSTM, resulting in a promising evaluation score. The interpretability of models like LSTM is complex when compared with baseline solution such as Xgboost. To summarize, all of the methodologies utilized in this thesis aid in the prediction of critical alarms before they occur, and this work will be used as a baseline for future Ericsson improvements in this area.

Bibliography

[1] C. Bertero, M. Roy, C. Sauvanaud and G. Tredan, "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection," 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2017, pp. 351-360, doi: 10.1109/ISSRE.2017.43.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in Proc. Interspeech, vol. 2, p. 3, 2010.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Proc. Advances Neural Information Processing Systems, 2013, pp. 3111–3119.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," J. Mach. Learn. Res., vol. 12, pp. 2493–2537, Aug. 2011

[5] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the ACM Conference on Computer and Communications Security, 2017. doi: 10.1145/3133956.3134015.

[6] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in IJCAI International Joint Conference on Artificial Intelligence, vol. 2019-August, 2019. doi: 10.24963/ijcai.2019/658.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, vol. 2017-December, 2017.

[8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, vol. 1, 2019.

[9] P. Korvesis, S. Besseau and M. Vazirgiannis, "Predictive Maintenance in Aviation: Failure Prediction from Post-Flight Reports," 2018 IEEE 34th International Conference on Data Engineering (ICDE), 2018, pp. 1414-1422, doi: 10.1109/ICDE.2018.00160.

[10] Gutschi, C., Furian, N., Suschnigg, J., Neubacher, D., Voessner, S., 2019. Log-based predictive maintenance in discrete parts manufacturing. Procedia CIRP 79, 528–533. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.

[11] X. Gu, S. Papadimitriou, P. S. Yu, and S. P. Chang, "Online failure forecast for fault-tolerant data stream processing," in IEEE 24th International Conference on Data Engineering, April 2008, pp. 1388–1390.
[12] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, "Automated it system failure prediction: A deep learning approach," in IEEE International Conference on Big Data. IEEE, 2016, pp. 1291– 1300.
[13] Y. Yuan, S. Zhou, C. Sievenpiper, K. Mannar, and Y. Zheng, "Event log modeling and analysis for system failure prediction," IIE Transactions, vol. 43, no. 9, pp. 647–660, 2011.
[14] D. R. Cox and D. Oakes, Analysis of survival data. CRC Press, 1984, vol. 21.
[15] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto, "Online failure prediction in cloud datacenters by real-time message pattern learning," in IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2012, pp. 504–511.
[16] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An Online Log Parsing Approach with Fixed Depth Tree, IEEE International Conference on Web Services (ICWS), 2017
[17] AWSOM-LP: An Effective Log Parsing Technique Using Pattern Recognition and Frequency Analysis, 2021, v1
[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," CoRR, vol. abs/1603.02754, 2016. arXiv: 1603 . 02754. [Online]. Available: http : //arxiv.org/abs/1603.02754.
[19] C. Olah, "Understanding lstm networks," Colah's Blog, Aug. 27, 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 04/26/2022).
[20] LONG SHORT-TERM MEMORY Neural Computation 9(8):17351780, 1997 available: http://www.bioinf.jku.at/publications/older/2604.pdf
[21] Distributed Representations of Sentences and Documents arXiv:1405.4053 2014
[22] Visualizing Data using t-SNE availability: https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Py
[24] L. S. Shapley, Notes on the n-Person Game II: The Value of an n-Person Game, 1951.
[25] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-tointerpreting-model-predictions.pdf.
[26] https://radimrehurek.com/gensim/models/doc2vec.html

A

Appendix 1

Figure A.1: Training/validation F1 score for LSTM many-to-many model

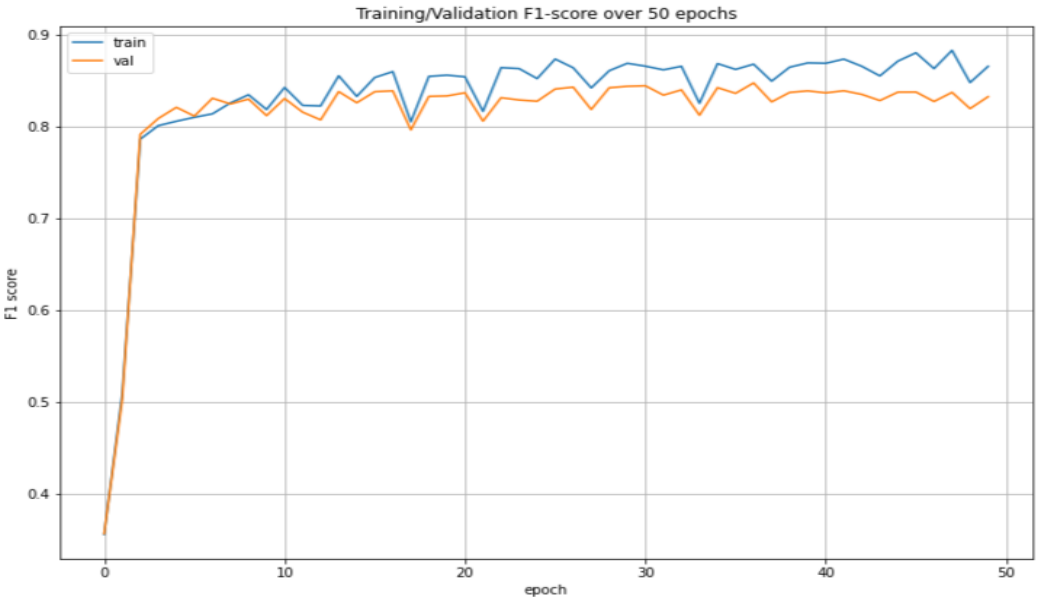


Figure A.2: Training/validation loss for LSTM many-to-many model

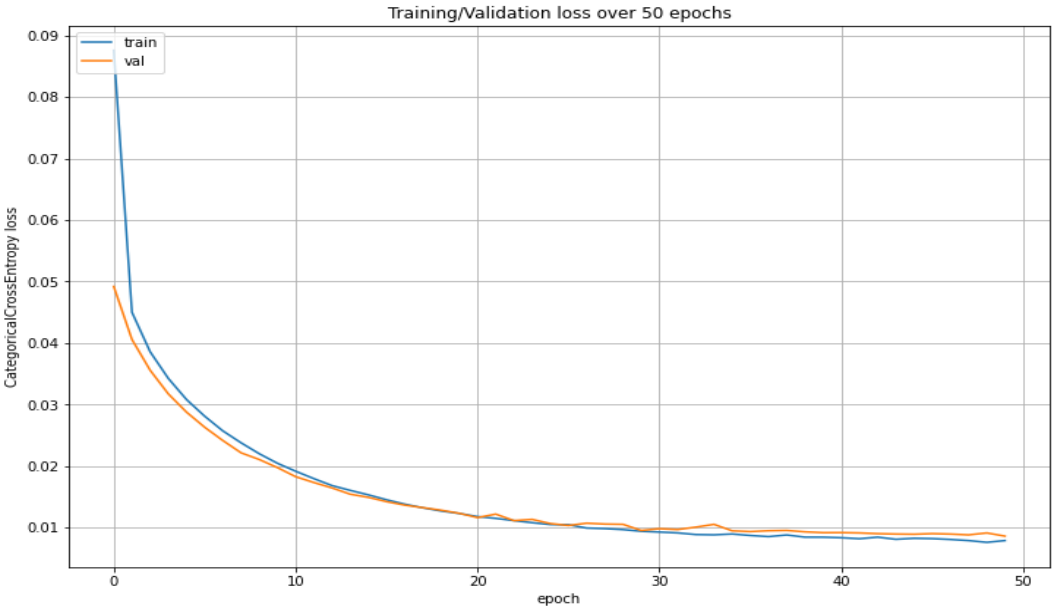


Figure A.3: t-SNE representation of doc2vec embeddings for radio fault and raised alarms