Online Anomaly Detection on Streaming Log Data Using Hierarchical Temporal Memory

Mohanavelu, Senthil Kumar

Award date:
2022

Department of Mathematics and Computer Science
Data Mining Group

Online Anomaly Detection on Streaming Log Data Using Hierarchical Temporal Memory

Master thesis

Author

Senthil Kumar Mohanavelu

Graduation supervisor

Dr. Mykola Pechenizkiy

Graduation co-supervisor

Dr. Stiven Schwanz Dias

September 28, 2022

Abstract

Complex software systems generate large amounts of data in the form of log messages with millions of messages generated each hour from a single software system. As systems grow larger and more complex, so does the information generated by such systems. Moreover, changes and updates in software introduce newer additional log messages along with with changes to existing messages. In a way, log messages constantly evolve and change over time with every single update or patch to the software.

Detecting anomalies in log messages can help prevent unexpected downtime, security threats and abnormal behaviour ensuring high availability and reliability of the software to end users. This is extremely important from a business perspective as every enterprise or company uses distributed software services in one way or another. An unexpected downtime might lead not only to financial losses but also to customer dissatisfaction.

While several log anomaly detection models exist, they do not satisfy all the requirements that are needed from a continuously learning, adaptive, and online anomaly detection model. In this thesis, the possibility of using a continuous online learning model, the Hierarchical Temporal Memory (HTM) model, is investigated. Several different approaches to feature extraction and encoding are presented and evaluated in the context of the HTM model. Results indicate that the proposed solution is capable of anomaly detection on the publicly available Hadoop Distributed File System (HDFS) log dataset. The thesis summarizes the benefits of using a HTM model while comparing its drawbacks against existing state-of-the-art deep learning models.

Contents

Chapter 1

Introduction

1.1 Motivation

Analysis of logs opens the door to detecting anomalies and diagnosing systems. Complex software systems and large businesses generate over 10 million log messages every hour from thousands of systems [3]. It is vital to analyze these log messages on the go to detect anomalies and issues in order to prevent failure and ensure high availability and reliability of machines. The high velocity and volume of log data render it impossible for an expert to manually analyze every single log message. Log messages are also not local to a single machine, complex software systems are often made up of multiple servers and nodes each generating logs in parallel. It is extremely difficult if not infeasible for a single developer to view and investigate this high volume of log data as it is being generated. Furthermore, with every addition of a new server or node, a new software, and with every new patch or update, new logs are bound to be introduced to the ecosystem. Automation of log analysis and anomaly detection can ease the human workload, save manpower and ensure that the machines are performing to their fullest. From a business perspective, the benefits are straightforward and multifold. Online anomaly detection not only helps save costs by identifying anomalous machines and software executions, but also helps assure the quality of services to the consumers. A consistent and powerful anomaly detection model will ensure the high availability and reliability of machines for its users.

Therefore, there is a need for an anomaly detection model that can detect anomalies in real-time logs as early as possible. Additionally, logs in practical cases do not contain labels and with the variety as well as the volume of log data it is not feasible for a developer or an expert to manually label these logs for analysis. Keeping these requirements in mind, the anomaly detection model needs to be capable of the following:

1. Unsupervised learning to learn from unlabelled data.
2. Continuous learning to ensure the introduction of new types of log messages from software updates and patches does not affect the performance of the model.

3. Perform online anomaly detection on streaming log data.

The following section explores three research questions that aim to develop a solution to address the points made in this section.

1.2 Research Questions

With this motivation in mind, the broad question is “Is it possible to develop a robust solution that is capable of detecting anomalies in real-life online streaming log data using the HTM model?" Taking into account the requirements described in the previous section, the following research questions are formulated:

1. How do different feature extraction methods affect the performance of the HTM model?
2. How do different encoding methods affect the performance of the HTM model?
3. Is the HTM model suitable for online anomaly detection on log data?

1.3 Literature review

Several approaches to log anomaly detection exist at the moment. Most approaches involve parsing logs to extract log templates followed by prediction or classification of the log templates [32], [31], [30] and [10]. Work under log anomaly detection can be broadly classified under three different categories: Supervised, Semi-supervised and Unsupervised.

1.3.1 Supervised methods

Several supervised approaches exist, most of which involve extracting count vectors by assigning indexes to each log template. Log templates are representative templates of log messages such that multiple log messages can originate from the same log template. These templates are assigned IDs and a log count vector is created by counting the number of occurrences of such templates. Models such as Support Vector Machines (SVM) [15] and Logistic Regression (LR) make use of log count vectors to make binary classifications on whether a log is anomalous or normal. Decision Trees have also proven to be quite good at detecting anomalies reaching precision and F_1 scores of upto 0.99 and 0.98 [4]. However, supervised approaches such as these require large amounts of labelled data and are severely affected by noise in the training data such as mislabelled logs. Furthermore, the models fail to utilize the semantic information of the log messages and solely rely on log indexes and count vectors. Most of the information contained in the logs is not utilized during the model classifications.

Deep learning techniques such as Convolutional Neural Networks (CNN), an neural network architecture primarily used in image recognition, has recently been adapted to solve anomaly detection in logs based on its capability to detect spatial local patterns [18]. Here, log templates are encoded as integers and grouped together

to form log sequences based on a common identifier for log sequences. The integer log sequences are then padded with zeros or trimmed to fit the fixed-dimensional input size of the CNN network. The network makes predictions on each log sequence as either anomalous or normal. The CNN model specifically reaches high performance on the benchmark HDFS dataset with a F_1 score of 0.98. However, the model is severely limited to the quality and quantity of the dataset and requires large amounts of labeled data for the model to be accurate [14]. Additionally, this approach fails to once again make use of semantic information contained in the messages

LogRobust [32] is another supervised deep learning technique that improves on earlier supervised deep learning models by incorporating semantic embeddings through a pre-trained word2vec FastText model for the log templates. The embeddings are then weighted using TF-IDF before being passed onto the attention-based Bidirectional Long Short Term Memory (LSTM) model The model detects anomalies by making classifications using the LSTM network. LogRobust's performance is impressive, with scores of F_1 up to 0.99 on the HDFS dataset. However, as the model performs classifications using an LSTM, it requires large amounts of labeled data that includes both anomalous and normal data. Furthermore, the model is sensitive to noise in training data where mislabelled logs hinder the performance of the model [14]

1.3.2 Semi-Supervised

Deeplog [9] is a semi-supervised deep learning approach that, once again, makes use of LSTM networks. As in earlier approaches, logs are converted to log templates and assigned indexes and are then grouped together as log sequences either using an identifier in the log message or through fixed time windows. The LSTM model is trained on log data that follows normal execution paths in order for the model to learn the execution paths of normal tasks. An anomaly is detected by comparing the sequential prediction of the LSTM with the ground truth message. Deeplog requires very little training data, and additionally, the authors provide a method to incrementally update Deeplog over time to ensure that the model adapts to new unseen log messages. However, the method involves a domain expert manually labeling false positives as normal messages and updating the weights.

LogAnomaly[22] builds a log anomaly framework that aims to utilize the semantic meaning of logs. The authors take advantage of language learning models to extract semantic information from logs by using a word2vec model called template2vec. New unknown log templates are handled by simply matching the new template to the closest template vector created during the training stage. While the model performs really well on the HDFS and BGL datasets, it still requires frequent retraining to stay up to date. Since new log messages simply match existing vectors, there is a varying loss of semantic information that hinders its performance.

1.3.3 Unsupervised methods.

[30] discusses a Principal Component Analysis (PCA) based method where the logs are grouped together by identifiers such as session IDs or by sequence IDs. A message count vector is generated for each sequence by simply counting the number of log-keys that appear during the session. Each column in the vector corresponds to a particular log template and the value indicates the number of times that log template appears in the session. The authors then use PCA to project the vectors onto a different coordinate system to detect anomalies. This method produces results with high precision and high recall on the HDFS dataset while remaining fairly straightforward and simple. However, the method is purely offline and requires the entire log dataset to detect anomalies. Furthermore, the method assumes that the logs can be grouped in sequences based on an identifier which might often not be possible based on the type of logs generated by a computer system.

Invariant mining [17] is an interesting approach to the problem that is based on program invariants. The approach focuses on finding linear relationships in the log messages by mining program invariants. Here, log messages are grouped by program variables, and message count vectors are formed for each log grouping. Once all possible invariants are extracted from the dataset, an anomaly can be detected by simply finding a sequence that violates the discovered invariants. The approach produces results that are comparable to state-of-the-art on Hadoop and CloudDB datasets. However, finding invariants requires a large enough dataset that contains historical data. Furthermore, the method is completely offline and cannot be directly used to detect anomalies for online streaming log data.

LogCluster [16] is another unsupervised approach based on clustering log messages in which event count vectors are generated from log sequences simply by counting the number of times a specific log template occurs in a sequence. The event count vectors are then classified as anomalous or normal using aggloremative heriarchial clustering techniques.

1.3.4 Summary

Various supervised approaches such as SVM, decision trees, and logistic regression have been proven to achieve good results on anomaly detection. However, they require large amounts of training data as well as labels, which are often hard to come by since labels need to be manually created by a domain expert for log data. Similarly semi supervised approaches such as Deeplog and LogAnomaly require non anomalous data for training whereas unsupervised approaches assume that anomalies are sparse and normal logs constitute the majority of log messages. Unsupervised deep learning approaches have shown great strides in recent years but most models perform offline anomaly detection and are quite sensitive to unseen data or unseen log sequences.

Deeplog and LogAnomaly produce good results on benchmark datasets while requiring little data to achieve their best performance and additionally, both models

provide methods to deal with unseen log messages. However, none of the online models are capable of continuous online learning as deep learning models such as CNN and LogRobust need to be retrained on the new data and models such as PCA, decision trees and SVM need to be refit on the entire dataset. Deeplog provides a method for online learning but requires the knowledge of a domain expert to label false positives and requires retraining of the model with the new data.

In short, the performance of existing approaches have been proven to be satisfactory on stationary datasets. However, most models do not have the capability of adapting to unseen log messages or online learning and the ones that do such as Deeplog and LogAnomaly require both constant re-trainings as well as labelled data. Additionally, at this point in time no existing method is capable of continuous online learning on log data to perform anomaly detection.

1.4 Thesis approach

In order to solve the problem of anomaly detection in logs as a real world problem, it is vital to consider the practical challenges involved. The developed solution not only needs to display high accuracy in detecting anomalies online but also ensure that the model is able to continuously learn and adapt to updates and patches in the software system that result in the addition of unknown log messages. For the same reasons, the thesis explores the idea of employing Hierarchical Temporal Memory (HTM) model, a model designed to replicate the neocortex, to detect anomalies in log messages.

HTM has already been proven to produce state-of-the-art results in anomaly detection of time series data [1]. However, current literature only tackles data that is either numerical or categorical. This is primarily because the HTM model is only able to process a very specific type of input: Sparse Distributed Representations (SDR) which are simply binary arrays that have more than 90% of their bits set to 0. Conversion of scalar and categorical data to SDRs is well defined [26] whereas translating log messages to SDRs is not explored at the moment. This thesis primarily focuses on investigating several different feature extraction and encoding methods to find a suitable way to represent log messages as sparse representations. The work then shifts towards making use of the HTM model to perform anomaly detection on a benchmark HDFS dataset.

The work implements over eleven different approaches toward encoding log messages as SDRs and finds that the combination of sentence embedding and scalar encoding techniques produces the best results out of all. Another approach of generating sparse vectors using an autoencoder shows promising results as well. The HTM model produces results that do not currently beat state-of-the-art in performance. However, the model is able to reach its best performance learning on the fly after assimilating very little streaming data.

1.5 Outline of the thesis

Chapter 2 provides background information on the HTM model and SDR representations, two concepts that dominate the majority of the thesis. The chapter also introduces well-established concepts including autoencoders and hashing algorithms. Chapter 3 provides a brief description of the problem statement as well as formalizes it. Chapter 4 provides the solution methodology and discusses the different encoding and feature extraction techniques. The following chapter explains the experimental setup and the corresponding results while trying to answer the three research questions. Finally, Chapter 6 provides a discussion on the results of the thesis and possible future work to improve the results from the HTM model.

Chapter 2

Theoretical Background

2.1 Hierarchical Temporal Memory

The Hierarchical Temporal Memory (HTM) is a model devised and proposed by Jeff Hawkins and Numenta in 2011 [11] and is primarily inspired by the Neocortex of the human brain. The HTM model primarily accepts temporal input in the form of SDRs and makes a prediction on the next possible data point in the sequence in the form of an SDR. The model learns temporal relations in the input sequence and continuously updates its weights with every new input it encounters and with every prediction. Learning is based on the difference between the predicted output and the original input that arrives. Through continuous learning, the model adapts to changes in the incoming data.

The HTM model consists of regions that are arranged in hierarchies where regions perform the most basic memory and performative tasks. Regions are allowed to communicate with other regions at the same level as well as with regions higher and lower in the hierarchy. This concept of regions is borrowed from the biological understanding of the human brain where the neocortex is classified into different regions based on where they receive input from and where they send their output. Figure 2.1 displays the compartmentalization of the human brain and the way information is processed at different stages in an hierarchical manner.

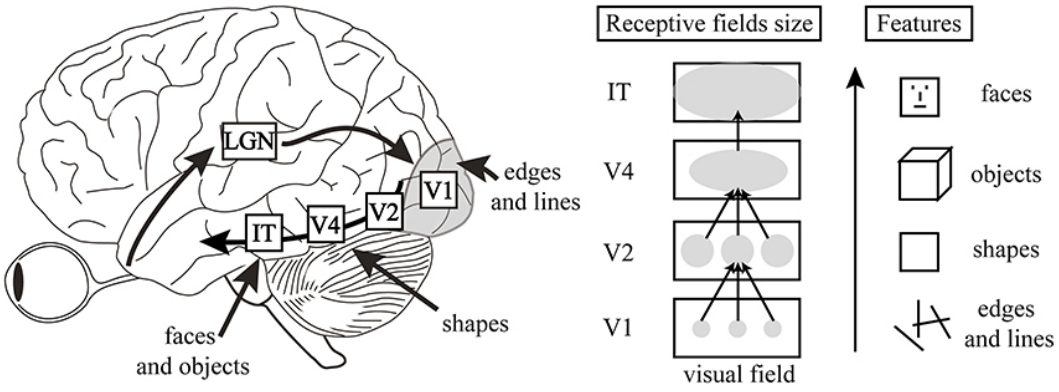

Figure 2.1: Illustration depiction the biological inspiration of the HTM model from [20].

The following subsections describe two of the major components of the HTM model: The Spatial Pooler and the Temporal Pooler.

2.1.1 Spatial Pooler

The Spatial Pooler (SP) is the first major part of the HTM model and is thoroughly described in the paper [5]. The SP learns a one to one mapping of the input SDR space to another sparse binary representation of minicolumns. An minicolumn can be thought of as an column of cells in the SP region. The Spatial Pooler consists of a number of minicolumns where each minicolumn is connected to different bits in the input space. These synaptic connections are called potential connections and contain weights called permeance which regulate the establishment and pruning of of the corresponding connections following a Hebbian learning approach. They are called potential connections in the sense that they only feedforward the input space when the permeance of the connection is above a certain threshold, and when it is under the threshold, no feedforward takes place.

The feedforward value of each synaptic connection of an SP minicolumn is calculated by multiplying the permeance with the value in the input space. Therefore, for an SP minicolumn, the feedforward value is the sum of all connection values and in a way, this measures the number of overlapping bits with a minicolumn. An SP minicolumn is set to active when the feedforward value is in the top 2% of values. The permeance of the connections that contribute to the value is strengthened while the ones that do not contribute is weakened. Through this learning method, similar input in the input space will fire similar minicolumns in the SP. Figure 2.2 shows an illustrative example of minicolumns being set to active based on the input space.

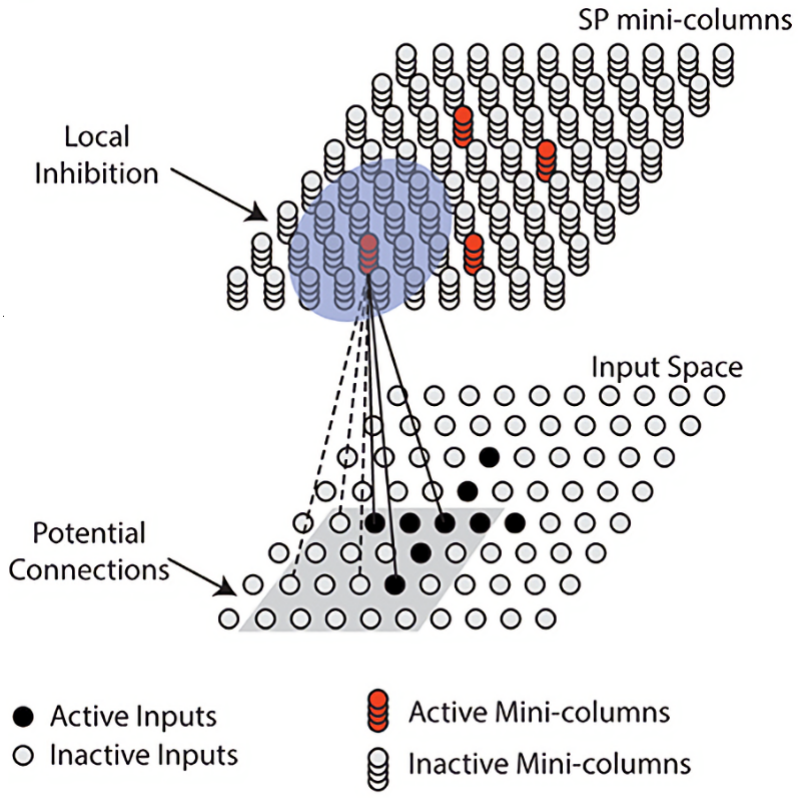

Figure 2.2: Working of the spatial pooler adapted from [5]

In the figure, the grey area represents the potential pool of input space that a single SP mini column is connected to. Here, the black lines are connections to active bits in the input space while the dashed lines are connections to inactive bits. As discussed earlier, the synaptic connections of the black lines are strengthened while the connections of the dashed lines are weakened. In the figure, the blue area on the SP indicates the local inhibition region where only one minicolumn is set to active based on a winner-take-all rule. Through this inhibition rule of minicolumns, the SP continues to maintain the required sparsity.

2.1.2 Temporal Pooler

While the SP produces an encoding that represents the value in the input space, the Temporal Pooler (TP) is responsible for representing the input along with the context with which it arrives. It is essential to recall that the SP produces an encoding that contains the active minicolumns. Each minicolumn is made up of a number of cells that can be present in three different states - active, inactive, and predictive. Furthermore, each cell has synaptic connections to other cells in the same region and each connection once again has its own permeance value. By setting different cells active within the same minicolumn, the TP is able to represent the same input based on different contexts. Furthermore, similar to the SP maintaining sparsity through local inhibition of minicolumns, the TP uses local inhibition of cells to

maintain sparsisity. Figure 2.3 provides an illustration of how the TM works to learn contextual inputs.

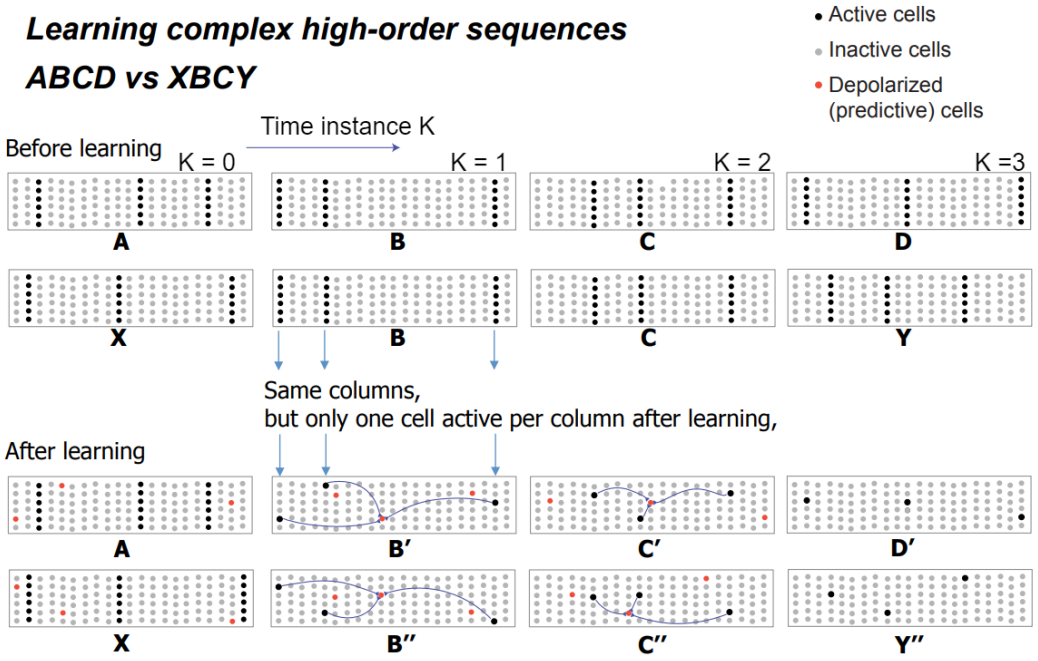

Figure 2.3: Illustration of sequence learning by the Temporal Pooler from [1]

The figure illustrates the learning process and the predictions made by a temporal process for two input sequences **ABCD** and **XBCY** containing the common subsequence **BC**. Here each column represents a minicolumn, while black dots and red dots represent active and predictive cells whereas grey dots indicate inactive cells. The minicolumns are set to active by the SP as explained in the previous subsection and here, **A** can be represented by the SDR $[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$ and $\mathbf{X} = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$. The TM produces output SDRs where only one cell in a minicolumn is set to active with the rest of the cells set to inactive. With the help of the figure as an example, the working of the TP can be summarized through three steps as follows:

1. At the arrival of an input at time instant k the TP checks every active minicolumn and turns “predictive cells” into “active cells”. If no "predictive cells" exist in an active column as in the case of a new and unseen input, all cells are set to active and this process is called "bursting". In the figure, the first rows show the bursting process where the model encounters the two sequences for the first time. Note that all the cells in a single minicolumn are set to active since no predictive cells exist at this time instant k. When predicive cells exist, local inhibition takes place since only the predictive cells of an active minicolumn are set to active while the rest are set to inactive. In a way, the predictive cells locally inhibit other cells from firing in the same minicolumn.

2. At the same instant k, the TP iterates through every active cell and feedforwards the input to connected nearby cells. If the value crosses a threshold, the

connected cells are set to the predictive state ready for the next input at $k+1$. In the figure, red cells indicate predictive cells set at a particular time instance. The predictive cells are calculated through lateral connections between cells of different minicolumns. The synaptic connections shown in the figure indicate the active cells that contribute to the calculation of the predictive cells for the next time instance. Since these connections are based on the cells active, the same input can be represented differently for different contexts. In the figure, **B'** and **B"** fire the same minicolumns in the SP but have completely different active cells under the same minicolumn and thus, denoting the same input under two different contexts: **AB'C'D'** and **XB"C"Y"**

3. At the arrival of the next input $k+1$, the connections for correctly predicted cells are increased while incorrect predictions lead to the weakening of connections.

Through the concept of predicted cells, the temporal memory learns to represent the same input under different concepts. In short, active mini-columns represent the input value and active cells represent the context of the input.

The output of the TM is a binary sparse array with active bits representing active cells of the mini columns. In the default algorithm, the HTM model consists of 2048 mini columns with 32 cells each. This results in a binary array of size 65536.

2.2 Sparse Distributed Representations

Sparse distributed representations are large binary arrays where a small number of bits are set to 1 and the remaining majority is set to 0. Sparse representations resemble the functioning of the neocortex wherein the activations of neurons are spread out sparsely. The bit values of 1 and 0 can be thought to correspond to active and inactive neurons, respectively. SDRs have emerged as a way to represent information in a way similar to that of the human brain. Typically SDRs have thousands of bits with just around 1% to 2% of the bits set to active. Individual bits on their own do not represent any specific labels or value but rather convey semantic meanings. This implies that SDRs representing similar information will hold active bits in similar locations. Therefore, a high overlap of bits indicates a higher level of similarity and vice-versa.

2.2.1 Properties of SDRs

SDRs have a few unique properties that make them ideal for various machine learning tasks.

1. SDRs display high capacity: The capacity of an SDR can be calculated by counting the number of unique combinations possible with the arrangement of 1 and 0 bits. For an SDR of size n with a fixed number w of 1 bit, this can be calculated as w combinations from n as folllows:

$$\left(\begin{array}{c} n \\ w \end{array} \right) = \frac{n!}{w!(n-w)!}.$$

For the standard SDR size of $n = 1024$ and $w = 40$, the number of possible combinations amounts to $1.46 \cdot 10^{72}$. This allows for plenty of different values to be encoded in an SDR of just 1024 bits. By adjusting the size n and controlling the sparsity through w, the capacity can further be increased or decreased to accommodate all possible values of the input space.

2. SDRs are not prone to noisy data: Similarity in the case of SDRs is calculated through an overlap score and not traditional methods such as Euclidean distance or Hamming distance. The overlap score is the count of overlapping 1 bit between the two arrays and can be defined as follows:

   Let $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$ denote two sparse binary vectors with at most w active bits, the overlap score is.

$$S_{overlap} = \sum_{i=1}^{n} \left[\mathbf{a}[i] \wedge \mathbf{b}[i]\right], \tag{2.1}$$

   where the Iverson bracket which takes as an argument any logical proposition Ψ is defined as

$$[\Psi] = \begin{cases} 1, & \text{if } \Psi \text{ is true} \\ 0, & \text{otherwise.} \end{cases}$$

   For two vectors, this is exactly the same as using the AND operator. Two SDRs are said to be exact matches if all their bits match. By using a threshold in conjunction with the overlap score, the sensitivity towards noise can be adjusted. While this may introduce false positives, note that the error bound climbs very slowly. For example for $n = 2048$ and $w = 40$, even with the addition of 14 noise bits, the error rate is less than 10^{24}.

3. SDRs allow for the representation of multiple values: A combination of values or patterns can be represented by simply performing the UNION operation over multiple SDRs. With sufficient sparsity, multiple SDRs can be combined without much loss of information or error. For example, two SDRs $\mathbf{a}$ and $\mathbf{b}$ and be fused into SDR $\mathbf{c}$ such that, $\forall i \in \{1, \ldots, n\}, \mathbf{c}[i] = \mathbf{a}[i] \vee \mathbf{b}[i]$.

4. SDRs are computationally efficient: The sparseness and the binary values of the SDR allow for easy computation required for various tasks. AND and UNION operations are fairly simple and calculating the overlap score and matching SDRs is also undemanding.

2.2.2 Encoding

The HTM model only processes information in the form of SDRs and in order to make use of the HTM model for classification and prediction, it is necessary to first encode the input data into an SDR. The paper [26] discusses several different strategies to encode various types of data ranging from single-valued integers to GPS coordinates. When encoding data to SDRs the following properties should be ensured.

1. Similar datapoints should result in similar SDRs or in other words, two similar data points should be encoded as SDRs with a high overlapping score. For

example, two similar SDRs **a** and **b** will need to have a high $S_{overlap}$ as described in Equation 2.1

2. The encoding from the input space to the SDR space needs to be bijective i.e. $I \longleftrightarrow S$ and should always result in SDRs of the same dimensions or size n.

3. The encoded SDRs needs to have the same sparsity or in other words, the same proportion of active bits w to toal number of bits n for every data point.

For an example of a simple numerical encoder, SDRs can be generated by assigning bits to overlapping ranges of numbers. Consider a SDR of size $n = 20$ and $w = 3$ with the range of values being $[0, 23]$. The first bit is set to active for values ranging from $[0, 3]$, the second bit is set active for ranges $[1, 4]$, and so on. Figure 2.4 displays a SDR of size $n = 20$ and $w = 3$ representing the number 7.

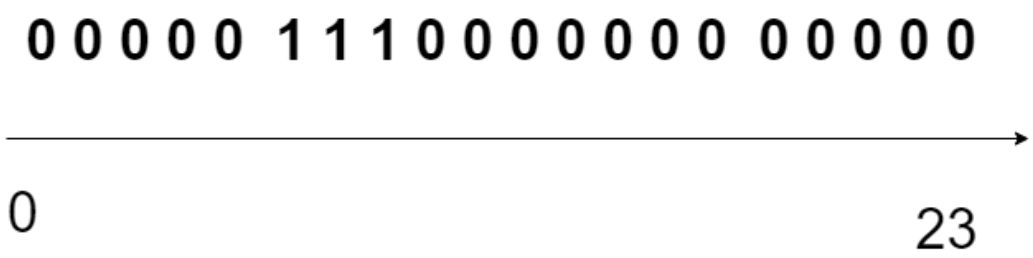

Figure 2.4: Illustration of a simple numerical encoder.

Another example of a categorical encoding is shown in Figure 2.5. Here, the first 5 bits are assigned to the label "Dog" and the last 5 bits are assigned to the label "Cat".

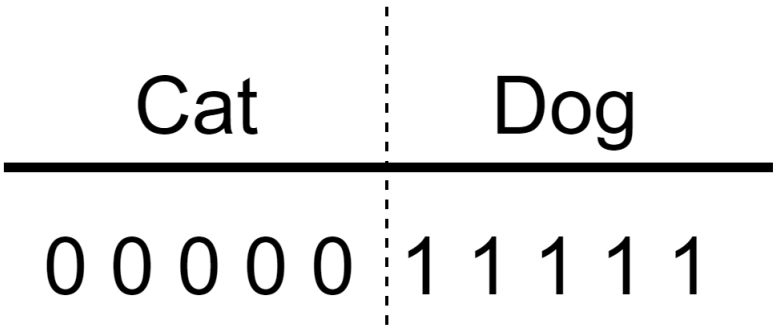

Figure 2.5: Simple SDR encoding for two disjoint categories.

Note that in order to encode categories, it is not necessary to have disjoint sequences of active bits. In order to encode the days of the week, one can simply assign an integer to each day $[1, 7]$ and use the numerical encoder.

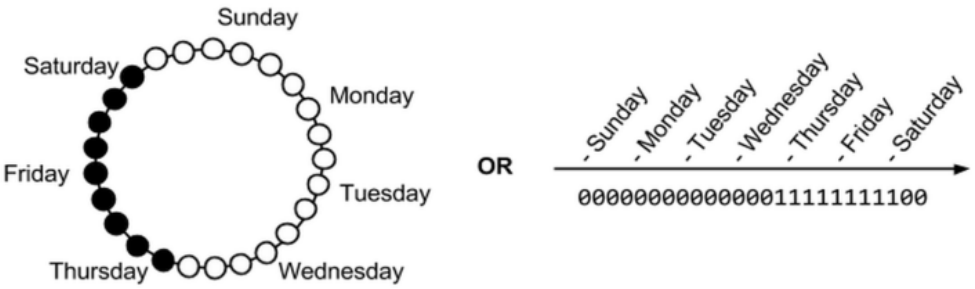

Figure 2.6: A simple SDR encoding for days of the week [26].

2.3 Autoencoders

Autoencoders are artificial neural networks that encode the data from the input space into a hidden latent space and reconstruct the original data by decoding the data in the latent space [2]. Formally, the autoencoder consists of two primary neural networks – an encoder and a decoder. The encoder tries to learn useful representations of the input space by converting an input vector in the input space to a higher or lower dimensional vector in the latent space whereas the decoder reconstructs the original input vector from the latent space representation. Formally, the encoder E encodes the input vector $\mathbf{x}$ to a latent space representation $\mathbf{z}$ where $\mathbf{z} = E(\mathbf{x})$ and the decoder D tries to reconstruct the input from the latent variable such that $\mathbf{x}' = D(\mathbf{z})$ where $\mathbf{x}'$ is the reconstructed input. Figure 2.7 provides an illustration of a traditional auto encoder structure.

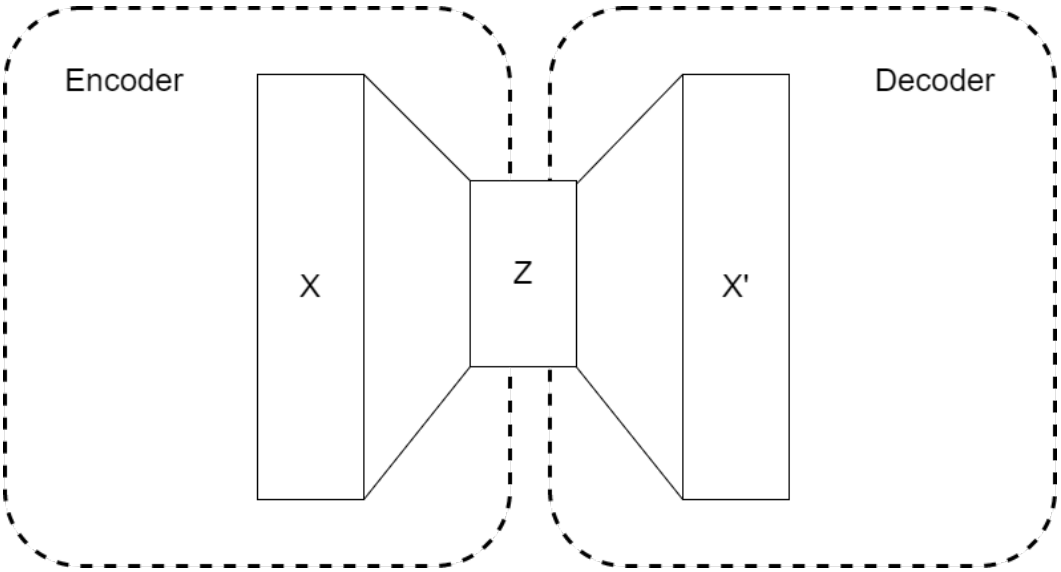

Figure 2.7: Schematic representation of the encoder-decoder architecture

The parameters of the auto encoder are optimized by comparing the reconstructed vector with the original vector by often considering the squared error loss function:

$$\mathcal{L}\Big(x, D\big(E(x)\big)\Big) = \|\mathbf{x}' - \bar{\mathbf{x}}\|_2^2$$

where $E(\mathbf{x})$ and $D(E(\mathbf{x}))$ denote the encoder and decoder outputs and x and x' represent the original and reconstructed input.

Autoencoders have been shown to be very useful in the case of dimensionality reduction by using an encoder to map the higher-dimensional input space to a lower dimensional latent space [13]. Another variation of autoencoders is the sparse autoencoder where sparsisity in the latent space is ensured through regularization. Sparsisity has been shown to improve the performance of various tasks such as classification and clustering. Furthermore, sparse encoders have been proven to aid in performing tasks such as speech recognition [7] and image classification [28]. One such sparse autoencoder is the K-Sparse Autoencoder [19] where in sparsitiy is maintained by trimming the output in the hidden layers. After the activation layer in the encoder phase, only the top k activations are allowed to pass through and the remaining activations are set to 0.

2.4 Dimensionality reduction

Principle Component Analysis (PCA) is a widespread method for reducing dimensionality. First proposed in [25], PCA is a statistical method that aims to represent high-dimensional data with dimensions d on a low dimensional space with principal components m where $d > m$. PCA essentially involves finding linear combinations of the d features that can explain the maximum variance. These linear combinations form the principal components m. It is important to note here that PCA is a orthogonal transformation and as such the principal components m have zero correlation between each other. The principal components are calculated through eigenvalue decomposition of the covariance matrix of the input dataset. The covariance between two column vectors can be calculated as follows :

$$\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})\,(y_i - \bar{y})}{n}$$

where $\bar{X}$ and $\bar{Y}$ denote the mean of the respective columns and n is the number of samples and $\mathbf{X}$ and $\mathbf{Y}$ are the column vectors in the input space. This results in a $d \times d$ square covariance matrix $\mathbf{M}$. The eigenvalues can then be calculated as:

$$\det(\mathbf{M} - \lambda\mathbf{I}) = 0$$

where $\mathbf{I}$ is the identity matrix and the roots of the equation provide the eigenvalues. The eigenvectors corresponding to the m largest eigenvalues are then used to transform the original higher n-dimensional to a lower m dimensionsal space.

Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) is a newer technique for dimensionality reduction that is based in Riemannian geometry [21]. The theory involves learning the manifold structure of the high-dimensional data and then approximating a equivalent one in low dimensional

space. It works through two major steps, the first step involves learning a fuzzy topological structure of the high-dimensional data. UMAP does this by building a connected weighted graph containing all the points in the dataset. Density of points is preserved by ensuring edges belonging to closer points contain higher weights and vice-versa. With a weighted graph constructed in the high dimensional space, the second step is to project the graph while preserving its structure onto the required lower dimensional space. UMAP ensures that the low dimensional topological structure is as similar as possible to the original high-dimensional structure. More information on the working of UMAP can be found in the original paper [21].

Chapter 3

Problem Statement

The problem as explained in the previous chapter is focused on detecting anomalies in system log messages. More formally, let

$$\mathcal{L} = (l_1, l_2, l_3, \ldots, l_n) \tag{3.1}$$

be a sequence of log messages such that $\exists l_i \in \mathcal{L}$ where $0 \leq i \leq n$ and l_i is anomalous. The goal of anomaly detection is to accurately label all such l_i as anomalous and the rest as normal. A log message is said to be anomalous if it shows deviation from normal process flow or any undesirable behavior. Additionally, the model needs to be capable of online unsupervised learning to ensure adaptability to changes in log messages with the introduction of software changes or updates.

Let $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_n)$ be the vector representation of the logs $(l_1, l_2, l_3, \ldots, l_n) \in \boldsymbol{\mathcal{L}}$ and let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ be the SDR encoding of the vectors where $\mathbf{x}_j$ is the sparse encoding of $\mathbf{v}_j$. The HTM model takes in the sequence of SDRs $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1})$ to make a prediction for the next log message in the sequence based on the history of events where the model output $\hat{\mathbf{x}}_{n|n-1}$ is a prediction of the HTM model. Lastly, the predicton $\hat{\mathbf{x}}_{n|n-1}$ is compared with the ground truth $\mathbf{x_n}$ to determine if it is an anomaly or not.

The problem can then be narrowed down to finding a suitable approach to adapt the HTM model to perform anomaly detection on natural language logs. This translates to finding an effective method of creating a sparse representation of the log message data. Let $\mathcal{I}$ be the input space of the vector representations $\mathbf{V}$, then the encodings $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ are produced by an encoding function $f : \mathcal{I} \to \{0, 1\}^m$ such that

$$\mathbf{x} = f(\mathbf{v}; m, w)$$

where m and w are the size of the SDR and the number of active bits, respectively. Note that the following must hold for any arbitrary vectors $\mathbf{x}, \mathbf{x}' \in I$.

$$\begin{aligned} &\forall \mathbf{x}, \mathbf{y} \in \mathcal{I}, \mathbf{d}_{\mathcal{I}}(\mathbf{x}, \mathbf{y}) \geq \mathbf{0} \\ &\forall \mathbf{x}, \mathbf{y} \in \mathcal{I}, \mathbf{d}_{\mathcal{I}}(\mathbf{x}, \mathbf{y}) = \mathbf{d}_{\mathcal{I}}(\mathbf{y}, \mathbf{x}) \\ &\forall \mathbf{x} \in \mathcal{I}, \mathbf{d}_{\mathcal{I}}(\mathbf{x}, \mathbf{x}) = \mathbf{0} \end{aligned} \tag{3.2}$$

where $d_{\mathcal{I}}$ is a distance function over the input space such that $\mathcal{I} \times \mathcal{I} \to \mathbb{R}$. Additionally let $a_1, a_2, \ldots, a_k$ where $a_i \in \{0, 1\}$ such that $1 \leq i \leq k$ be the elements of a binary

array $\mathbf{A}$ of length k. Then a counting function $C(\mathbf{A})$ is defined as:

$$C(\mathbf{A}) = \sum_{\mathbf{i=1}}^{\mathbf{k}} [\mathbf{a_i} = \mathbf{1}]$$

Let w be the number of active bits in the encoded SDR $\mathbf{x}$, then for an encoding $\mathbf{x}$ produced by $f(\mathbf{v}, m, w)$. The following stays true:

$$C(\mathbf{x}) = \mathbf{w}.$$

Finding a suitable encoder for the input space will allow for the training of the HTM model which can then be used to make predictions on the next log message in order to detect anomalies.

Chapter 4

Solution Approach

This chapter outlines first the HTM-based solution for the problem – online detection of anomalies in streaming log data and then details the proposed approach for the log feature extraction and encoding stages of the solution. The working of the HTM model and its usage has already been covered in Section 2.1. The following sections explore different feature extraction and encoding strategies and therefore cover multiple disjoint approaches. However, the HTM model remains the same irrespective of the different techniques used to encode the log messages as SDRs.

The workflow of the proposed solution can primarily be distributed into four stages as follows:

1. Parsing - Parsing the logs to retrieve log templates
2. Feature extraction - Extracting valuable features from the log templates
3. Encoding SDRs - Encoding the features in the form of usable SDRs
4. Anomaly detection - Using the predictions of the HTM model to detect anomalies
    a) Prediction - Use the HTM model to predict the next log message.
    b) Prediction assessment - Compute the prediction error using the predicted and actual log messages.
    c) Anomaly detection - Declare anomalies by thresholding an anomaly score based on the prediction error.

Figure 4.1 outlines the workflow of the overall solution from raw HDFS logs to detecting anomalies.

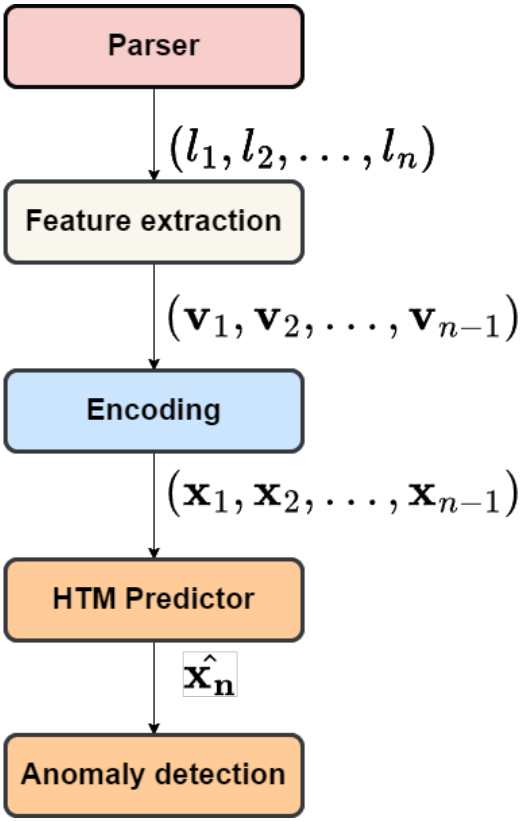

Figure 4.1: Workflow of the proposed solution

The rest of the chapter is split into sections based on the different stages of the proposed solution.

4.1 Parsing - Spell

The first stage of the proposed solution is parsing the raw HDFS logs. This is performed through the use of an existing log parser called Spell [8]. Figure 4.2 displays an outline of the parsing stage. The "log templates" are the primary fields that are extracted from the raw logs.

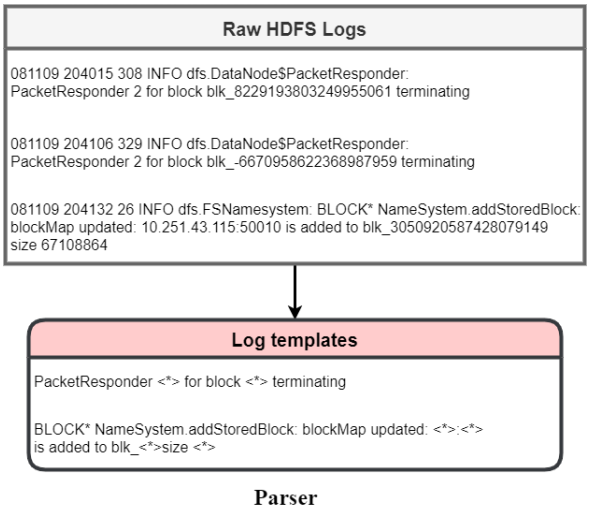

Figure 4.2: Outline of the parsing sage using Spell.

Spell is an online log parsing algorithm that works on the idea of identifying the Longest Common Subsequences (LCS) to extract log templates where $LCS(L, J)$ retrieves the longest common subsequence between L and J . The algorithm works on the principle that log messages are predefined by the developer in the source code and are usually of the format of a given template with one or more variables or parameters.

For example, an example log message could be

```
printf(’’ Error at line \%d’’, lineid)
```

and possible resulting log messages could include

1. `Error at line 131`
2. `Error at line 21`

Here, the log message:

```
‘‘Error at line <*>’’
```

remains the same irrespective of the line id at which the error occurs while <*> can be replaced with different line IDs. The underlying aim of Spell is to identify the structured templates such as the example above that exist in the code. The algorithm identifies such templates through the idea of LCS which works as follows.

Algorithm 1 Parse log messages $L = (l_1, l_2, \ldots, l_n)$ into log templates

Require: Log messages $L = (l_1, l_2, \ldots, l_n)$
1: Initalize empty map $LCSmap$
2: Let $LCS(a, b)$ be a procedure that returns the longest common subsequence $\in (a, b)$
3: **for** $l \in (l_1, l_2, \ldots, l_n)$ **do**
4: $\alpha = 0$
5: **for** $m \in LCSmap$ **do**
6: **if** $\alpha < |LCS(l, m)|$ **then**
7: $\alpha = |LCS(l, m)|$
8: **end if**
9: **if** $\alpha > \tau$ **then**
10: Map l to existing template in $LCSmap$
11: **else**
12: Create new template in $LCSmap$
13: **end if**
14: **end for**
15: **end for**

1. An empty map, LCSmap, is first initialized and a threshold τ is set by user where the threshold determines the the length of LCS required for a new log template to be generated.

2. Incoming log messages are split into tokens and the tokenized message is compared to existing templates in the LCSmap.

3. If $max\left(LCS(new, LCSmap)\right) > \tau$, then the new log entry is assigned to the existing log template. If $max < \tau$, the new log entry is stored as a new log template. If no log templates exist in LCSmap, the message is stored as a new log template.

The threshold τ is usually set at 0.5 implying that logs with more than half of the tokens as parameters are not expected. Since the algorithm keeps track of existing log templates and any new log entry simply needs to be tokenized and compared with the existing map, the usage of Spell for online log parsing and log template extractions is extremely easy. The Spell algorithm runs on unstructured log data resulting in structured log data that includes log templates. All existing logs would have been generated using one of the log templates discovered by Spell.

Table 4.1 displays raw logs that have been retrieved from the HDFS dataset.

No	Raw log
1	**081109 203615 148** INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
2	**081109 203807 222** INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-6952295868487656571 terminating
3	**081109 204005 35** INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864
4	**081109 204015 308** INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_8229193803249955061 terminating
5	**081109 204106 329** INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-6670958622368987959 terminating
6	**081109 204132 26** INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.115:50010 is added to blk_3050920587428079149 size 67108864

Table 4.1: Raw log messages from the HDFS dataset

The resulting event templates that are extracted by Spell are shown in Table 4.2. Every single log message in the raw logs must have originated from one of these two log templates.

Event Id	Log Template
E10	PacketResponder <*>for block blk_<*>terminating
E6	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*>:<*>is added to blk_<*>size <*>

Table 4.2: Event templates extracted from the raw logs by Spell on the HDFS dataset

Using Spell to parse the entire HDFS log dataset with over 11M messages results in just around 50 log templates and they are enough to describe the entire HDFS log dataset. The section below discusses the different ways features can be extracted from the log templates. Figure 4.3 displays the occurrences of the various log templates.

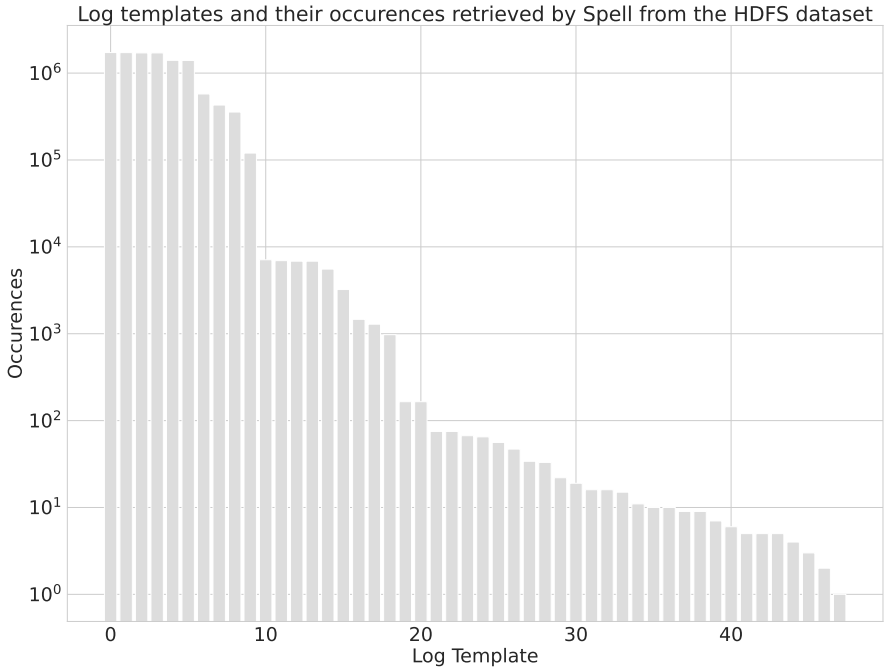

Figure 4.3: Occurrences of the various log templates retrieved by Spell from the HDFS dataset.

As evident from the Figure 4.3, the log messages are not equally distributed across different templates. Therefore, simply considering each template as its own category with equal weights is not the most ideal approach and methods such as word embeddings and IDF based weighting will be able to extract more meaningful information from the message itself. The following sections explore the different feature extraction techniques as well as the different ways to encode data as SDRs.

4.2 Feature extraction

The previous section explored the use of a log parser to extract multiple different fields from log messages. For this section, feature extraction primarily concerns itself with the log templates that were retrieved by the parser. The aim is to extract meaningful features that are usable by the HTM model to detect anomalies.

One natural approach is to investigate Natural Language Processing (NLP) methods such as word embeddings and sentence embeddings. Additionally, information retrieval techniques that are often useful for clustering documents are investigated as well. Figure 4.4 shows the outline of the feature extraction process at this stage of the overall solution.

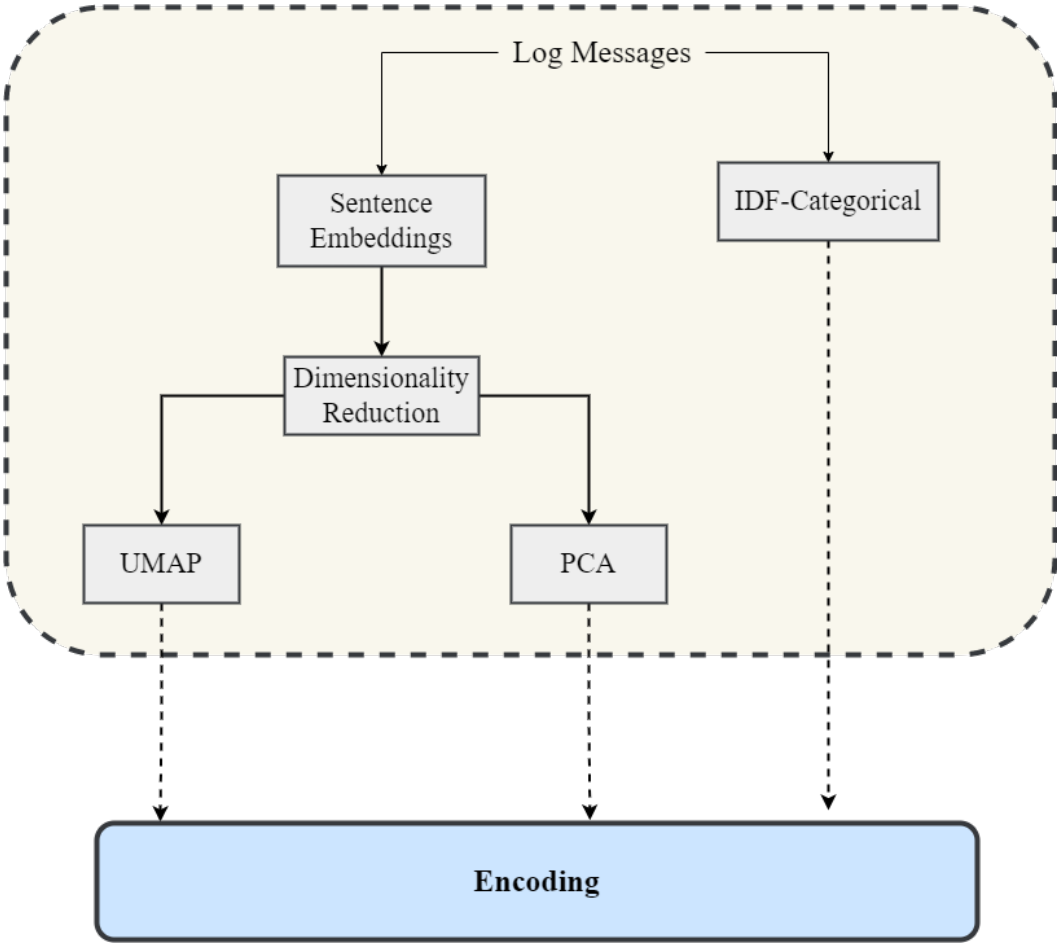

Figure 4.4: Outline of the feature extraction stage.

4.2.1 Sentence Embeddings

The methods discussed in this section use one or more methods to extract sentence embeddings from the log messages. Semantic information of the log messages can be obtained by converting the log templates and the list of parameters from the earlier log parsing stage into word embeddings.

Numerous word embedding and sentence embedding models exist such as Word2vec [23], template2vec [22], and transformer models such as Sentence BERT [27]. Word2vec and similar models encode each word or token into a specific vector irrespective of the context. This results in words that are spelled alike having the same encoding. For example, the word “bank” takes different meanings for the following sentences “Loan from a bank” and “Swim by the river bank”. In the context of logs, the messages are often full sentences and therefore, capturing accurate semantics and context is crucial.

Sentence BERT [27] a.k.a SBERT, on the other hand, encodes complete sentences as vectors of fixed dimensions. SBERT captures semantic and context-specific information from a sentence by conditioning on both left and right contexts in a sentence

as described in the original paper. As a result, the same word can be represented through different embeddings based on the location and context of the word.

There are plenty of pre-trained sentence transformer models that are publicly available through *huggingface* [1]. For log messages, the pre trained "cli-token-model" is utilized to convert the cleaned log templates to floating-point vectors with fixed dimensions of size 768. However, the HTM model handles data in the form of SDRs as explained in Section 2.1. In order to encode a SDR that contains the entire vector of length 768, an array of approximately size 102912 would be required by simple concatenating the bits belonging to SDR representation of each vector component. The number 102912 can be arrived at by considering that a single numeric value would ideally require 134 bits according to the Reference [26] discussing encoding strategies. Therefore, it is vital to reduce the dimensionaility of the word embeddings while still preserving the semantics and the distribution of the original data. The following section discusses different possible dimensionality reduction approaches.

4.2.2 Dimensionality

As discussed earlier, the HTM model is only able to handle input in the format of SDRs. The HTM library contains encoders designed by Numenta based on encoding techniques discussed in the Reference[26]. The existing encoders are capable of encoding multiple different data types including integer values, scalar data, categorical data, and even spatial data. However, before making use of the scalar encoders, it is important to understand that the HTM algorithm currently is not effective with data containing of more than five fields. The algorithm will end up requiring an extremely large number of data points to accurately learn temporal and spatial patterns.

A possible solution to this is to apply dimensionality reduction methods before passing the data to the encoders. Numerous methods exist that have been proven to be accurate at capturing information and meaning from a high-dimensional space and projecting it onto a low dimensional space. In the proposed approach, two of the most common methods are analyzed - PCA and UMAP.

4.2.3 Categorical IDF

Term frequency - Inverse document frequency (TF-IDF) is a common technique in information retrieval used to rank the importance of a word or term in a particular document belonging to a corpus. The TF-IDF of a term is calculated by combining two different statistical features. More specifically the TF-IDF is the product of term frequency (TF) and inverse document frequency (IDF) and is given by

$$TFIDF_{t,d} = TF_{t,d} \times IDF_{t,d}.$$

The first half of TF-IDF, namely the term frequency simply calculates the frequency of a term occurring in the document. While there are multiple different augmented methods to calculate the term frequency, the basic version that directly

[1] https://huggingface.co/

counts the number of occurrences and calculates the frequency by finding the proportion of the term occurring to the whole document is used here. The term frequency can therefore be given by

$$TF_{t,d} = \frac{total\ number\ of\ occurences\ of\ term\ t\ in\ document\ d}{total\ number\ of\ terms\ in\ document\ d}.$$

The second half of TF-IDF, involves the inverse document frequency which is calculated by first counting the number of documents containing the specific term and dividing the number by the total number of documents. The inverse of the value is the IDF value as the name suggests.

$$IDF_{t,D} = \frac{total\ number\ of\ documents}{total\ number\ of\ documents\ containing\ term\ t}.$$

In the proposed approach of encoding log messages, log templates are first extracted and each log template is considered as its own category. Here, the log sequence is considered to be a document and each categorical log template is considered to be a term in the document. Now, the IDF is calculated as follows

$$IDF_{l,D} = \frac{\text{total number of log sequences}}{\text{total number of log sequences containing log template } l}.$$

In the end this results in two features being extracted:-

- Categorical - Log template
- Scalar - IDF value for each log template

The features are encoded using a combination of a scalar encoder and a categorical encoder to be converted to SDRs before processing by the HTM model.

4.3 Encoding

This section describes the different encoding methods that are investigated to form SDRs from the extracted features of the previous section. Figure 4.5 displays the process at this stage of the overall workflow.

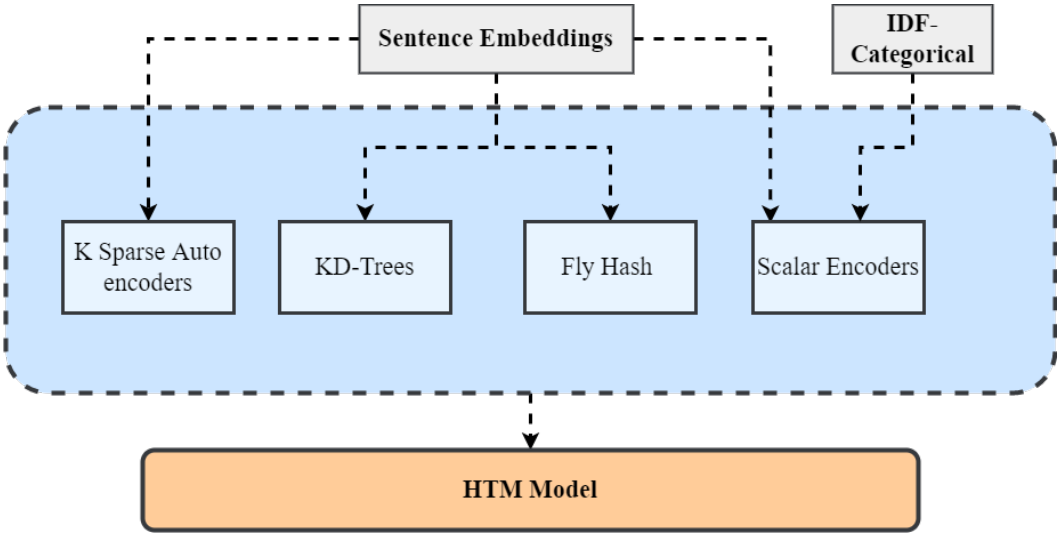

Figure 4.5: Outline of the encoding stage

4.3.1 Scalar Encoders

The Scalar encoder[26] is an encoding technique designed by Numenta to encode numerical and floating-point values. For a fixed size SDR of size n with width w, the data is split into fixed-size buckets depending on the range of the input data. The width w indicates the number of "ON" or "1" bits. Each bucket is denoted by assigned w bits as 1 and therefore a total of $(n - w) + 1$ buckets can be formed. This results in an SDR where numbers that fall into buckets that are closer to each other have more overlapping bits compared to values that fall in buckets that are farther away.

For example, consider $n = 8$, $w = 2$ and let the input space exist in the range $(1, 24)$. Figure 4.6 shows the resulting SDRs that are generated through the encoder when the inputs are integers 2 and 6.

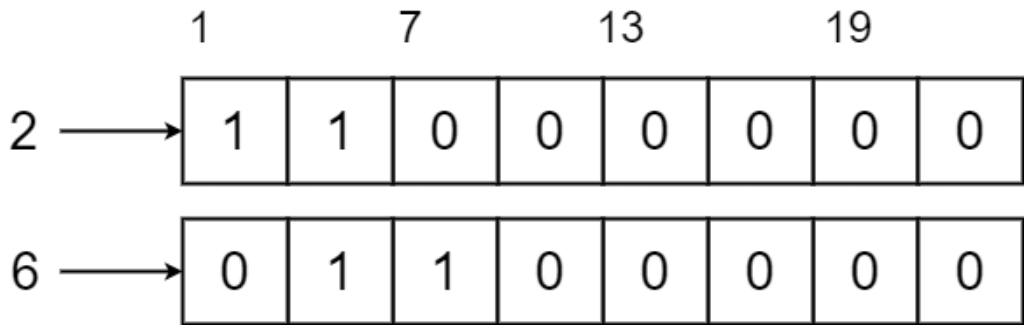

Figure 4.6: Illustrative example displaying how the numbers 2 and 6 are encoded when $n = 8$ and $w = 2$.

Here, it is trivial to note that the numbers 2 and 6 will have one overlapping bit while the numbers 2 and 8 will have 0 overlapping bits. For the proposed approach

the NuPIC [2] library provides an implementation of the Scalar encoder from Numenta. In the case of encoding multiple values or features that are numeric in nature, an SDR for each value is calculated and later concatenated to form a single large SDR.

4.3.2 K-D Trees

K-D trees are a way to represent k-dimensional points in an efficient manner that allows searching for nearest neighbours. The trees are binary and are constructed by iteratively splitting the hyperplane of each dimension based on the median point. Each node in the tree splits the points equally along one specific dimension.

As an alternative approach to scalar encoders which simply encode numeric data, the aim of using K-D Trees is to retain some of the spatial information of the input data. By capturing information from nearest neighbours, the resulting SDRs of close points in the higher-dimensional space will be similar while points farther apart will be dissimilar.

In this approach, K-D trees are constructed by first considering a fixed sample of data points. Each node of the K-D Tree is assigned to a bit in the SDR. In order to represent a new point, the closest n neighbors are retrieved from the K-D Tree and the corresponding bits in the SDR are set to 1. This ensures that the relative location of a point in K -D space can still be accurately encoded while still preserving some amount of uncertainty.

The steps involved in the construction are as follows Here, n is the required size

Algorithm 2 Calculate $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_m)$

Require: $0 < d < m$

Construct a KDTree using n points
Construct array $\mathbf{A}$ of size n
for $i \in (1, 2, \ldots, n)$ **do**
    $\mathbf{A}_i \leftarrow 0$
end for
Let $Nearest(\mathbf{x})$ be a procedure that returns the index of w nearest neighbours of $\mathbf{x}$
for $ind \in Nearest(\mathbf{V})$ **do**
    $\mathbf{A}_{ind} \leftarrow 1$
end for

of the SDR and w is the width or the number of 1 bits in the SDR.

4.3.3 Fly-hash encoder

The algorithm first described in [6] is explored as an alternate means to both encode higher-dimensional sentence embeddings in a lower-dimensional space as well as encode the data into a SDR. The algorithm is inspired by fruit flies' olfactory senses

[2] https://github.com/numenta/nupic

and is similar to locality-sensitive hashing. The proposed approach as described in [6] projects the input space onto a sparse higher-dimensional space by computing dot products against sparse binary vectors that are axis-aligned. Figure 4.7 provides an illustrative example.

Each bit in the sparse higher-dimensional space maps to only a select number of values in the input space. Since the projection vectors are binary, the dot product is the sum of all the corresponding values in the input space that map to a specific higher-dimensional space. The top 5% of the values are set to 1 in the sparse matrix and the remaining are set to 0. This results in a hashing space where data that is similar in the input space have similar resulting hashes or in this case similar SDRs. Algorithm 3 describes the exact working of the proposed encoding technique

Algorithm 3 Calculate $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_m)$

Require: $0 < d < m$

Let $\mathbf{V} \in \mathbb{R}^d$ be the feature vectors of the log message
Set k as the number of bits to set active and m as the size of the SDR
Set p the percentage of connections
for $i \in (1, \ldots, d)$ **do**
    for $j \in (1, \ldots, m)$ **do**
        $\mathbf{M}_{ji} \leftarrow 1$ with probability p
    end for
end for
$\mathbf{O} \leftarrow \mathbf{M} \times \mathbf{V}$
$TopK \leftarrow k$ largest elements $\in \mathbf{O}$
for $i \in (1, \ldots, m)$ **do**
    if $\mathbf{O}_i \in TopK$ **then**
        $\mathbf{O}_i = 1$
    else
        $\mathbf{O}_i = 0$
    end if
end for

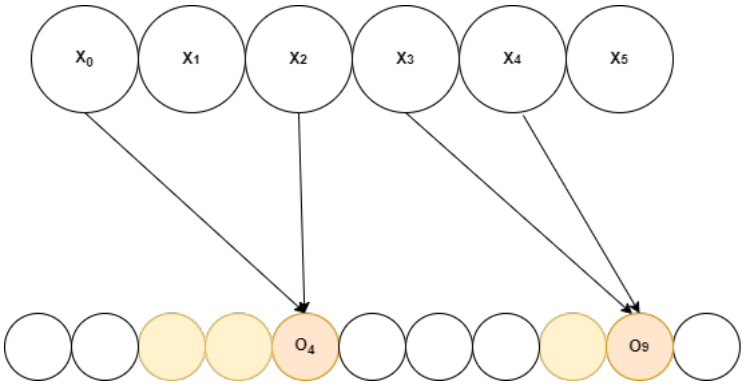

Figure 4.7: Illustrative example of how binary sparse projection works with a higher-dimensional hashing space.

In Figure 4.7, consider the input feature vector $\mathbf{X} = (x_0, x_1, \ldots, x_6)$ and the output sparse vector $\mathbf{O} = (o_0, o_1 \ldots, o_{10})$ where every node is mapped to a fixed number of values in the input feature vector. In this case two every single output node maps to two nodes in the input space. Therefore, every node in the output space contains the sum of two differet nodes in the input space. For example, $o_4 = [x_0 + x_2]$ and $o_9 = [x_3 + x_4]$ as shown in the figure. The nodes corresponding to the largest sum values are set to 1 and the remaining are set to 0. Since only the top 5% are set to 1, sparseness of the resulting higher-dimensional array is always maintained.

4.3.4 K-Sparse Autoencoders

As discussed earlier in Section 2.3, K-sparse encoders are based on autoencoders where only the top k activations are allowed to pass through in the encoder network. In the implementation of the encoder to encoder SDRs, it is vital that the resulting SDR is not only sparse but also binary. Therefore, the k-sparse autoencoder is modified to set the top k activations in the encoder stage to 1 while the remaining activations are set to 0 resulting in the required SDR.

In the case of the solution approach, sentence embeddings from the feature extraction stage are converted directly to sparse representations through a k-sparse autoencoder. The output at the end of the encoder network in the autoencoder is used as SDRs for the HTM model. The core idea of using an autoencoder is that the network would be able to learn to represent the high dimensional sentence embeddings in an SDR format of different sizes and sparsity. The size of the SDR can be controlled by simply controlling the size of the encoder network and the sparsity can be controlled simply by the k value of the autoencoder.

4.4 Real-time Anomaly detection

After the feature extraction and the encoding stages the messages are encoded as binary array representations in the form of SDRs, namely $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$. Note that the null hypothesis H_0 is that no anomaly exists. Conversely, for hypothesis H_1, $\exists \mathbf{x}_a \in \mathbf{X}$ such that $\mathbf{x}_a$ is anomalous where $0 \leq a \leq n$.

When the HTM model receives the input $\mathbf{x}_{i-1}$, it makes a prediction $\hat{\mathbf{x}}_{i|i-1}$ for $\mathbf{x}_i$ based on the current input. Once the model receives the input $_i$, it calculates the prediction error P_{err} and the anomaly likelihood. The prediction error gives information about the similarity of the predicted SDR and the ground truth SDR and is given by

$$P_{err} = 1 - \frac{\mathbf{x}_i \cdot \hat{\mathbf{x}}_{i|i-1}}{|\mathbf{x}_i|}$$

P_{err} is a scalar value and it is easy to note that the closer the value is to one, the less similar the two SDRs are or rather they share no similar "1" bits. A value closer to 0 indicates that the SDRs are almost identical in the placement of their "1" bits.

An anomaly is detected by first computing the prediction error for a new log message and classifying it as anomalous if the error crosses a pre-determined threshold

τ. In the case of log sequences where multiple logs are grouped together by a single identifier, the average of anomaly scores over the entire sequence is used for anomaly detection. Formally, the null and alternate hypothesis are

$$H_0 : x_i \ is \ normal$$

$$H_1 : x_i \ is \ anomalous$$

where we reject the null hypothesis H_0 when $P_{err} > \tau$ and accept the null hypothesis when $P_{err} \leq \tau$

Chapter 5

Results and Evaluation

5.1 Dataset

HDFS: The experiments are performed primarily on the HDFS dataset first introduced in [30] by Xu et al. The dataset contains logs retrieved from a Hadoop Distributed File System running on a private cloud environment resulting in over 11M log lines. Log sequences are associated with block ids with labels of normal or anomalous being assigned to each block id. The labels were created manually by a domain expert while following specific rules to assign anomalies.

BGL: The BGL dataset is an open-source log dataset presented in [24] that contains logs collected from the supercomputer Blue Gene/L. The dataset contains over 4M log lines with each log line labeled as either anomalous or normal. The dataset does not follow traces or sequences and therefore, log sequences are generated by applying a moving time window of 5 minutes. This results in 61840 log sequences, ranging from $1 - 3645$ messages in a single sequence. Furthermore, the dataset contains repeating logs that are grouped together with anomalous and normal logs appearing in chunks instead of individually. To make the dataset more suitable for the HTM model, repeating messages are trimmed so that every consecutive message is different from the previous one.

Table 5.1 displays the details of the two datasets. Note that anomalies in the case of HDFS are labelled by blocks where as BGL labels individual log lines as anomalies.

Datasets	**# of logs**	**# of anomalies**	**# of log sequences**
HDFS	$11,175,629$	16,838 (blocks)	575061
BGL	$4,747,963$	$348,460$(logs)	-

Table 5.1: HDFS and BGL datasets.

5.2 Metrics

Anomaly detection can be modelled as a binary classification problem wherein each data point is either classified as anomalous or normal. In order to measure the performance of the HTM model, standard metrics such as precision, recall, accuracy

and F-1 scores are used. To obtain these metrics the following terms are defined:

True positives (TP): Number of true anomalous data points that have been accurately classified as anomalous by the model.

False positives (FP): Number of data points that have erroneously classified as anomalous when they are originally non-anomalous points.

True negatives (TN): Number of true non-anomalous data points that have been correctly classified as non-anomalous by the model.

False negatives (FN): Number of anomalous data points that have been miss classified as normal by the model.

With TP, FP, TN and FN defined, the next step is to calculate precision, recall and the F_1-score.

5.2.1 Precision, recall and F_1

Precision is defined as

$$P = \frac{TP}{TP + FP}.$$

Intuitively, it can be defined as the model's ability to accurately predict anomalous points. This is easy to understand from the definition since P is directly the proportion of correctly classified anomalies to the total number of anomalies discovered by the model. A higher precision (close to 1) implies that the model rarely mistakes in pointing out anomalies whereas a lower score (close to 0) implies the opposite.

Recall is defined as

$$R = \frac{TP}{TP + FN}.$$

Recall provides information about the proportion of anomalies detected by the model. A higher recall value implies that the model was able to catch most of the original anomalies while a lower value implies that the model missed most of the anomalies existing in the original data. However, it is important to note that a model classifying every point as anomalous would score very high on recall but poorly on precision. Therefore, it is important to look at a combination of both metrics before deciding on the quality of the model.

F_1 *score* - The F_1 is defined as a means to combine both the information from recall as well as precision and is defined as the harmonic mean of the two as follows

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}.$$

5.2.2 Anomaly Score

The HTM model is used to calculate a raw anomaly score s_t based on the predictions of the temporal pooler where the score measures the proportion of mini columns that were incorrectly predicted by the temporal pooler of the model. The idea behind the formulation is that the HTM model would not be able to make a good prediction for unseen anomalous data and thus resulting in a majority of the minicolumns being unpredicted. This would result in a higher score whereas a data point that the HTM model is able to successfully predict would result in a lower score. The calculation of the anomaly score is provided as

$$s_t = \frac{|A_t - (P_{t-1} \cap A_t)|}{|A_t|},$$

where A_t is the active bits at the current timestep and P_{t-1} is the predicted bits by the temporal pooler from the previous timestep. Figure 5.1 illustrates an example of a good prediction and a bad prediction.

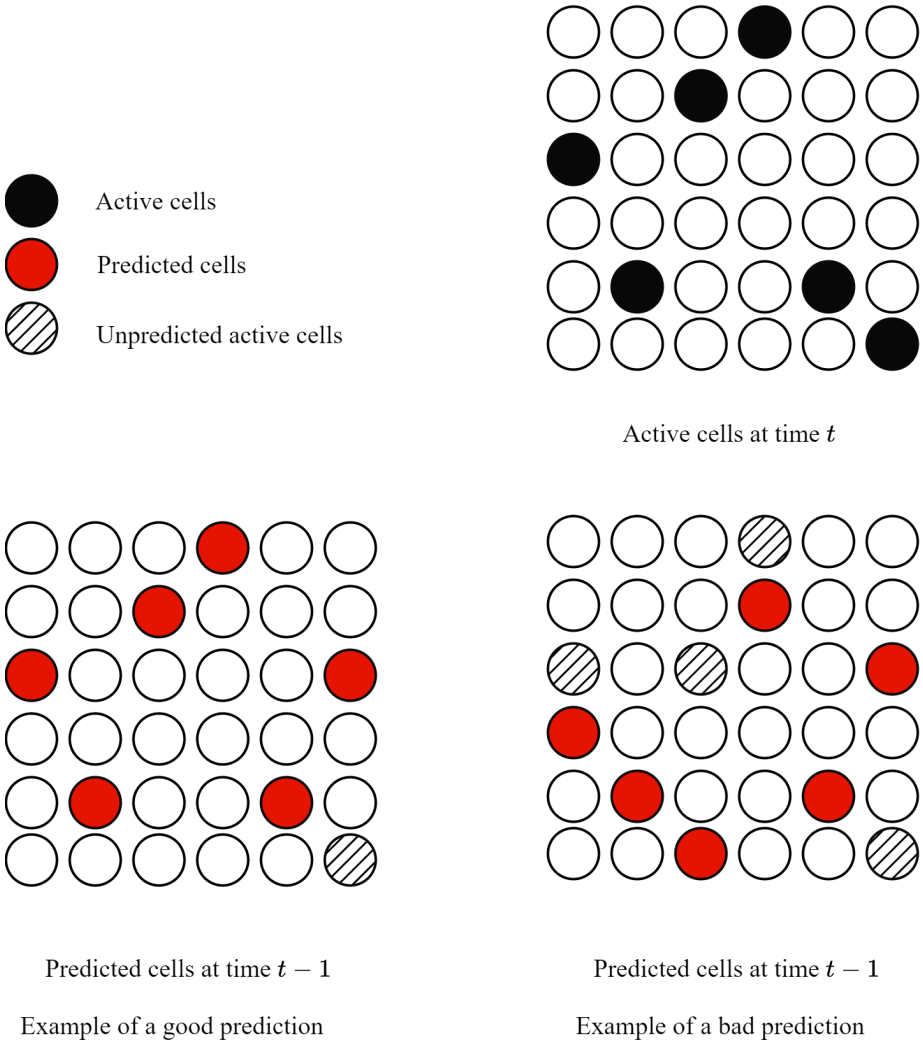

Figure 5.1: Illustrative example of predictions by the HTM model.

From the figure, it is easy to notice that a good prediction involves most of the active cells being correctly predicted indicating significant overlap where as a bad prediction implies the opposite.

5.2.3 Moving average precision

A moving window over log sequences is used to calculate the average precision as a measure of the performance of the model over time. Given a series of log sequences $\mathcal{SL} = (\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n)$, the moving average precision at log sequence $\mathcal{L}_i$ for a window of size m is the average precision of the model for the series $(\mathcal{L}_{i-m}, \mathcal{L}_{i-m+1}, \ldots, \mathcal{L}_i)$. Note that each log sequence $\mathcal{L} = (l_1, l_2, l_3, \ldots, l_k)$ is a sequence of log messages.

5.3 Experimental Methodology

In order to answer the research questions first proposed in Section 1.2, four different experiments are performed and inferences are drawn from the results. The following sections outline the experimental methodology as well as discuss the results from the experiments. The rest of the chapter is organized into sections based on the research question. Each section constructs an experiment to answer one research question followed by the results of the experiment while providing a short discussion elaborating on the inferences made from the results.

The experiments were run on a cloud instance with a Xeon® E5-2699 v4 cpu, 32GB of memory and 1x Tesla V100 GPU. The HTM model was built using the *htm.core* libary[1] which is a reimplementation of the *NuPIC* library[2]. The parameters of the HTM model used in the experiments can be found in Appendix A.

5.4 RQ1 - How do different feature extraction methods compare?

In Section 4 two different feature extraction methods were proposed: 1) Sentence Embeddings 2) Categorical IDF. The purpose of the experiment described in this section is to investigate the viability of both feature extraction methods for the purpose of anomaly detection by the HTM model. In order to test their viability, two HTM models are trained for anomaly detection: one that uses sentence embeddings and one that uses IDF weighted log key encodings. Additionally, in order to further analyze the impact of IDF weighting, a simple HTM model is trained that simply encodes different log templates as categorical datal. The performance of the three HTM models in terms of precision, recall, and F_1 score is compared to decide the most optimal feature extraction technique in the context of an HTM model.

[1] https://github.com/htm-community/htm.core
[2] https://nupic.docs.numenta.org/

5.4.1 Experimental Setup

IDF-Categorical: Log templates are extracted through the parser and arranged in log sequences based on block ids of the HDFS dataset. The IDF values for each log temmplate are calculated by using a 10% training sample using the TF-IDF vectorizer from sklearn library [3]. Log templates are mapped to IDF values while IDF values of new log templates discovered during online training are set to 0. Here, it is important to note that the HTM processes two input fields. The log key is encoded as categorical 'EventId' using a scalar encoder while the IDF value is encoded as a real value using a scalar encoder. Table 5.2 displays the parameters of the encoders.

Params	**EventId**	**IDF**
fieldname	*event*	*idf*
maxval	55	15
minval	0	0
n	600	130
type	*ScalarEncoder*	*ScalarEncoder*
w	21	21

Table 5.2: Parameters of the encoders for IDF-Categorical

Categorical: In addition to IDF-Categorical, a trivial approach of simply encoding log templates as category is compared against the other feature extraction techniques. Here, log templates are categorically encoded as SDRs using the built in encoders from NuPIC.

Sentence embeddings: Logs are processed to retrieve log templates which are then used to extract features through sentence embeddings. A pretrained SBERT model from *huggingface* [4] is used to encode each log message into a vector of size 768. The embedding is then reduced to a lower-dimensional input space (4 dimensions) through a dimensionality reduction method. The 4-dimensional vector is then encoded as an SDR using scalar encoders from *NuPIC*. Table 5.3 shows the details of the scalar encoders.

Params	**PC1**	**PC2**	**PC3**	**PC4**
fieldname	*PCA_1*	*PCA_1*	*PCA_1*	*PCA_1*
maxval	17	17	17	17
minval	−12	−12	−12	−12
n	250	250	250	250
type	*ScalarEncoder*	*ScalarEncoder*	*ScalarEncoder*	*ScalarEncoder*
w	21	21	21	21

Table 5.3: Parameters of the encoders for sentence embeddings

The choice of a dimensionality reduction method might impact the results and therefore, two of the most common and versatile methods(PCA and UMAP) are investigated to answer the sub-research question:

[3] https://scikit-learn.org/

[4] https://huggingface.co/sentence-transformers

- Does the choice of a dimensionality reduction method impact the performance of anomaly detection?

Dimensionality reduction through PCA and UMAP is implemented using the *sklearn* [5] library that is publicly available. The model parameters remain the same and both the methods reduce the 784 vector sentence embedding to 4 dimensions.

5.4.2 Results

Figure 5.2 displays the bar plot visualizing the performance of both the dimensionality reduction techniques.

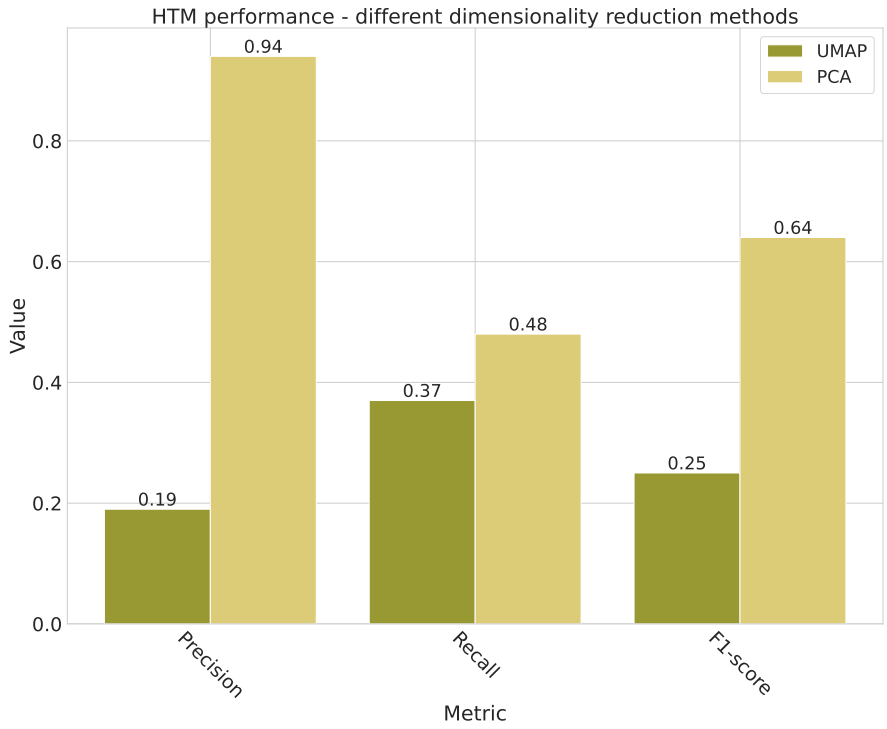

Figure 5.2: Performance of the HTM model using dimensionality reduction techniques based on PCA and UMAP.

From the figure, it is clear that the PCA performs significantly better than the UMAP. This could be explained by the way both techniques construct the lower-dimensional space. UMAP tries to find a mapping from a higher-dimensional space to a lower-dimensional space while preserving distance similarities where as PCA produces principal components that are orthogonal to one another. In other words, the 4-dimensional vector generated by PCA contains information in orthogonal axes while the mapping performed by UMAP might still contain related or overlapping information between mapped components. The HTM model when encoding as SDRs

[5] https://scikit-learn.org/

considers each dimension to be separate and can not learn relational information. The different dimensions are encoded into a single SDR which is then processed as a whole by the HTM model and as such, the HTM model is not able to explore any relational information between the components or dimensions. Therefore, the HTM model performs poorly in the case of UMAP compared to PCA.

Figure 5.3 displays the bar plot visualizing the performance of the three feature extraction techniques.

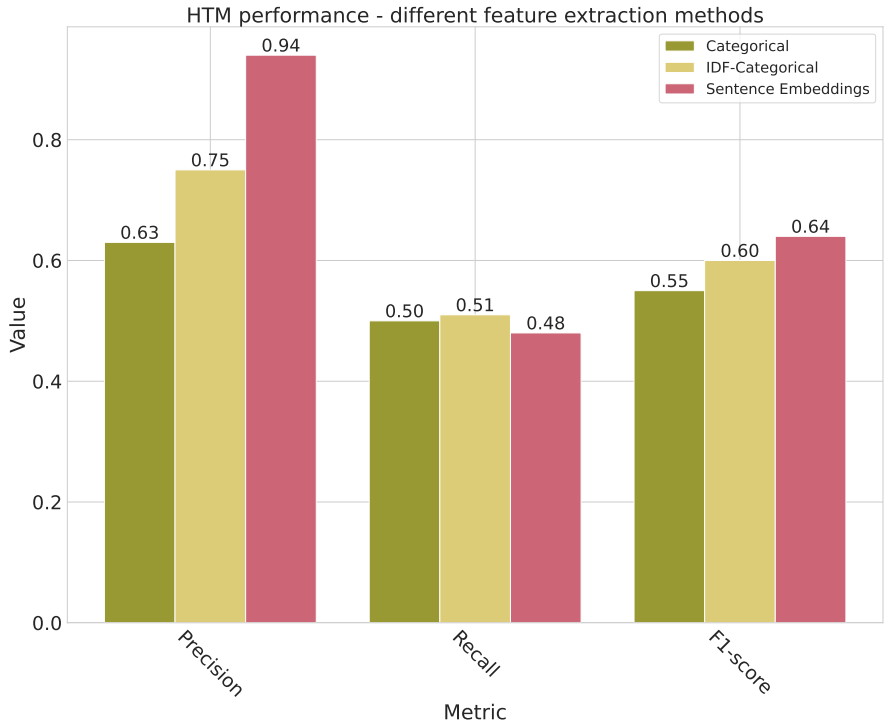

Figure 5.3: Comparision of HTM performance based on IDF-Categorical and Sentence embedding feature extraction techniques.

From Figure 5.3, it is evident that the sentence embedding technique clearly outperforms the IDF-based technique for anomaly detection. The difference in performance is most noticeable in the measurement of precision. The IDF-based method is only able to extract the relevance of a log key and none of the semantics of the message itself while the sentence embedding technique captures semantic information of the log message. By capturing semantic information, the sentence embedding technique is able to identify anomalies more accurately and thus lowering the amount of false positives (FP). In the case of IDF-Categorical and Categorical methods, the resulting SDRs at the HTM model are likely to have little to no overlapping bits. However, in the case of sentence embeddings, two similar log messages are likely to have overlapping bits. Therefore, when an unseen message is processed, the HTM model in the case of categorical and IDF-Categorical is more likely to produce a higher anomaly score. Whereas, the overlapping bits help the HTM model to relate

the unseen log message to previously seen messages.

Additionally, the recall of all the three feature extraction techniques are similar at around 0.48 for sentence embeddings upto 0.51 for IDF-Categorical. This implies the following:

1. Around 50% of the anomalies are identifiable by all three feature extraction techniques.
2. The difference in performance across precision and F_1 score can be primarily explained by the resulting overlapping bits under different techniques.

With the above results in mind, it is clear that different feature extraction methods produce different results. The combination of sentence embeddings and PCA clearly outperforms the other feature extraction that are proposed.

5.5 RQ2 - Does the encoding method impact the performance of Anomaly detection?

The previous research question focused on the feature extraction stage of the proposed solutions. This section focuses on the next stage - encoding the features to SDRs. The experiment in this section implements 4 different encoding methods to investigate the impact of performance that an encoding method may or may not have on the HTM model.

5.5.1 Experimental Setup

The setup includes the HTM model trained using SDRs generated by four different encoding methods: Scalar, K-d trees, FHE, and K - Sparse Autoencoders. Figure 5.4 provides an outline of how the different encoders process the input.

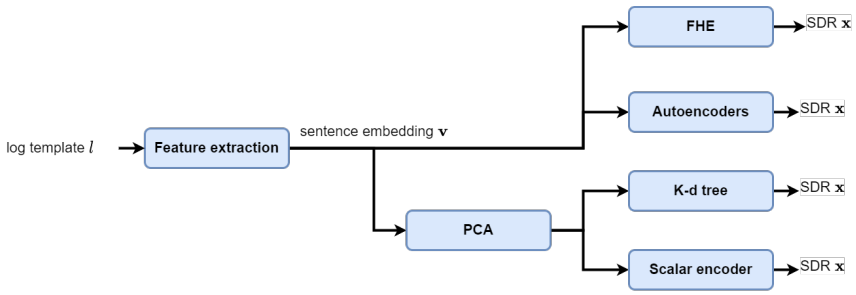

Figure 5.4: Pipeline of the different encoders

Scalar encoding: The sentence embeddings are first reduced to 4 dimensions through PCA which are then encoded as SDRs using the implementation provided by NuPIC. In this approach, each principal component has its own scalar

encoder resulting in 4 different SDRs each of size $n = 256$ and sparsity $w = 16$. The SDRs are then concatenated to form one single large SDR of size $n = 1024$ and $w = 64$

K-d trees: The K-d tree is constructed by using a sample of 1024 points from the input space of log embeddings. Each node in the K-d tree is assigned its own bit location in the resulting SDR and when an incoming log message is processed and its lower-dimensional embedding is extracted, the nearest 64 nodes in the K-d tree are retrieved and the bits mapped to the 64 nearest nodes on the SDR are set to 1. This results in an SDR of size $n = 1024$ and $w = 21$ that can then be processed by the HTM model's spatial pooler.

Fly hash encoder: Similar to previous methods, sentence embeddings of the log templates are retrieved by using a pre-trained sentence transformer. The embeddings of dimension 768 are then projected onto a sparsely distributed matrix using the fly hash algorithm as described in Section 4.3.3. The resulting SDR is of size $n = 1024$ with $w = 64$ and the HTM model is trained using a *PassThroughEncoder*.

Autoencoders: Here, a K-sparse autoencoder is first trained on sentence embeddings retrieved from 10% of the dataset to learn the encoding space. The K-sparse autoencoder converts the high-dimensional 768 sentence vector to a sparse representation of size 1024. The hyper-parameter of the autoencoder k is set to 21 and therefore, the encoding layer results in a binary sparse representation of size $n = 1024$ with $w = 64$ of its bits activated. This encoding is directly used as an SDR input to the HTM model through a *PassThroughEncoder*.

5.5.2 Results

Figure 5.5 plots the performance of the HTM model for different encoding methods used in the proposed solution.

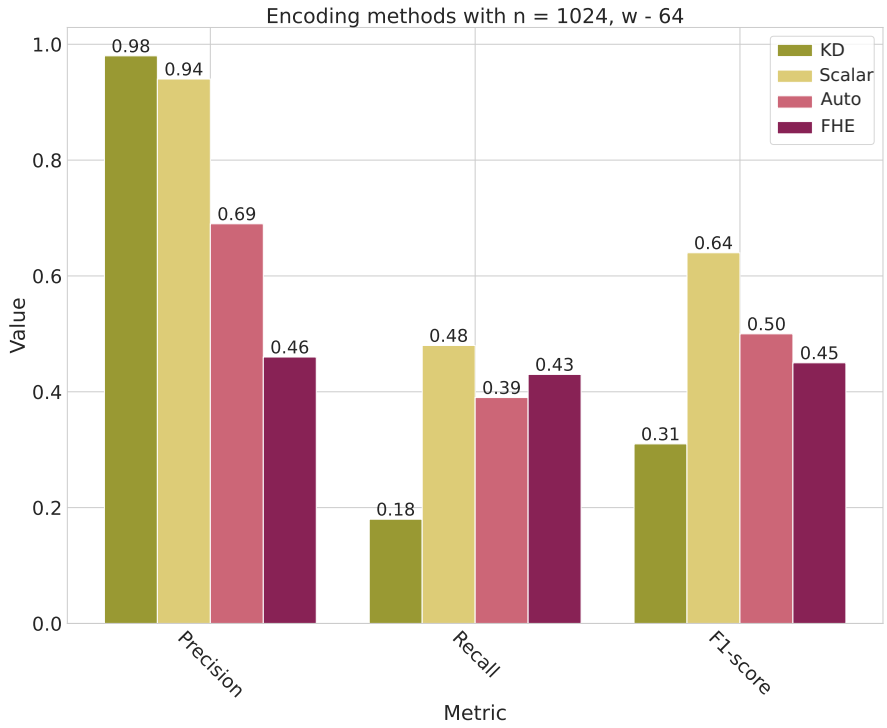

Figure 5.5: Bar plot of the performance of the HTM model for different encoding methods.

The bar chart indicates differences in performance between the different encoding methods. The scalar encoding produces the best F_1 score at 0.64 out of all the encoding methods with the highest recall at 0.48 and the second highest precision at 0.94. FHE and Autoencoders produce similar results across recall and F_1 scores while the precision of Autoencoders is significantly higher than that of the FHE encoding. The K-d tree approach results in near perfect precision at 0.99. However, the recall is the lowest at 0.18.

Both K-d tree and scalar encodings produces the least amount of false positives (FP) which can be inferred from their high precision values. This can be explained by the fact both these encoding methods make use of PCA to reduce dimensions of the sentence embeddings. Additionally, both methods have a clearer control over the number of overlapping bits. Similar PCA values are bound to result in larger number of overlapping bits between the SDRs. However, the K-d tree approach seems to produce a large number of false negatives as well. This could be explained by the way the approach constructs the K-d tree. The constructed tree has a large number of duplicate nodes as the number of unique log templates is much lower than the number of leaves.

The FHE and the K-sparse autoencoder approach directly convert the sentence embeddings to SDRs. This allows for lesser control over overlapping bits for similar values. While both the approaches have been shown to preserve distance similarities

to an extent, they perform poorly in the case of the HTM model for this particular problem of log anomaly detection. The FHE approach is unsurprising as the algorithm has previously been shown to produce poor results in converting word embeddings to sparse vectors [6].

In short, it is clear that the choice of an encoding approach can drastically affect the performance of the model in anomaly detection. The scalar encoding approach produces the best possible result in terms of F_1 score while FHE and autoencoders lag behind with similar results. The poor performance of FHE and autoencoders could also be the direct result of using log templates instead of log messages. FHE and autoencoders might be more capable in identifying subtle similarities and difference in log messages that are now lost due to the grouping of several different log messages to a single common log template. Finally, the results of the K-d tree approach are still inconclusive as it is unclear how the K-d tree might perform with a larger number of unique nodes.

5.6 RQ3 - Is the HTM model suitable for anomaly detection on log data?

This section designs three different experiments to explain the viability of the HTM model for anomaly detection from a business and practical perspective. More importantly, the experiments evaluate the model based on its ability to perform anomaly detection while ensuring its ability to adapt to evolving data on the go. The following subsection looks into one of the challenges that was encountered repeatedly while performing the experiments in this chapter. The remaining subsections continue to evaluate the performance of the HTM model for anomaly detection.

5.6.1 Additional challenge

One of the challenges that were encountered during the experiments was the processing speed of the HTM model. The model's recall time grows linearly with large datasets when online learning. To verify this, the speed of the model is investigated by plotting the time it takes to process each log message. For this experiment, the HTM model is used to perform anomaly detection on the HDFS dataset using the encoding method based on auto encoders. The choice for this particular encoding method is because the resulting SDRs are of size 2048 which remains the default value of a number of mini columns in the spatial pooler of the HTM model. The same experiment could have been performed using the K-d tree or FHE encoding method to arrive at the same results.

Since the logs are processed in sequences or blocks, the time taken for the SP, the spatial pooler module responsible for learning a mapping of the input data and the TM, the sequence learning module to process one log message or rather one SDR is calculated and plotted in Figure 5.6

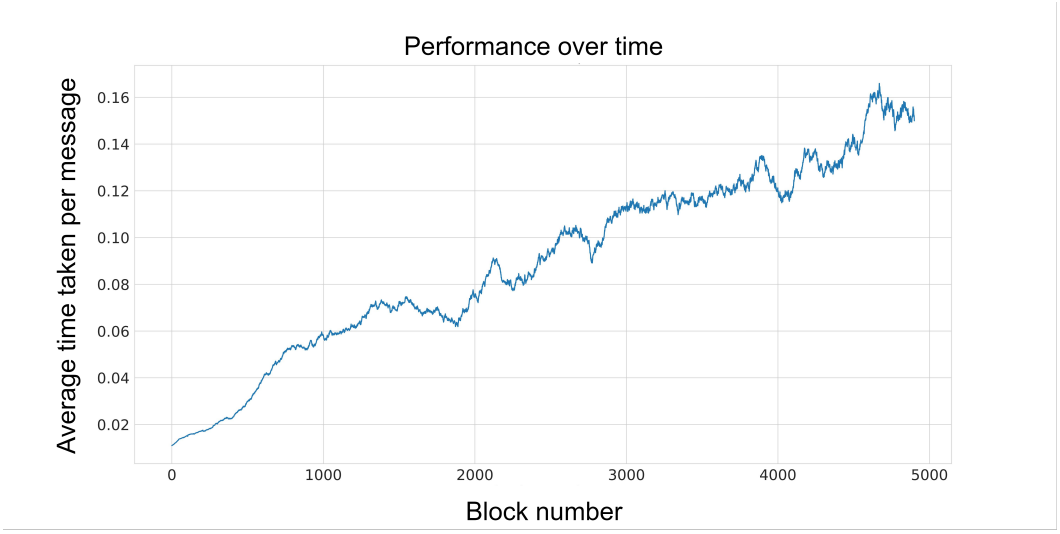

Figure 5.6: Average time taken per log message plotted over the first 5000 blocks with SDRs of size 2048.

From the figure, it is evident as the HTM model continuously trains over time, the processing speed of the model drastically reduces. The first block has an average processing time of around 0.02 seconds per message while block 5000 has an average time of 0.15 per log message. For reference, the time required to process the remaining 570061 blocks at a speed of 0.15 seconds is around 712 hours. Furthermore, this duration is calculated considering the average time required to be constant. Keeping practical aspects in mind, the rest of the experiments are performed on a sample dataset of the HDFS with around the first 100k log messages.

5.6.2 Performance on other datasets

The goal of this experiment is to evaluate the performance of the model on a significantly different dataset without standard log sequences. Since the HTM model is based on learning sequences that are temporal, the model is expected to perform poorly on datasets that do not follow sequences. To verify this, the dataset of BGL is chosen containing logs retreived from 128k processors of the BlueGene/L supercomputer. The dataset does not contain any standard log sequences that can be separated by block ids or instance ids. Instead, log sequences are generated by using a moving window model where each window size is determined by duration.

Experimental Setup: Features are extracted through sentence embeddings and PCA and encoded using scalar encoders which, as shown earlier, is the best possible configuration. The performance of the HTM model is measured using the same metrics of precision, recall, and F_1 score.

Results As expected, the performance of the HTM model is extremely poor with the BGL dataset shown in Figure 5.7 compared to the performance of the model on the HDFS dataset.

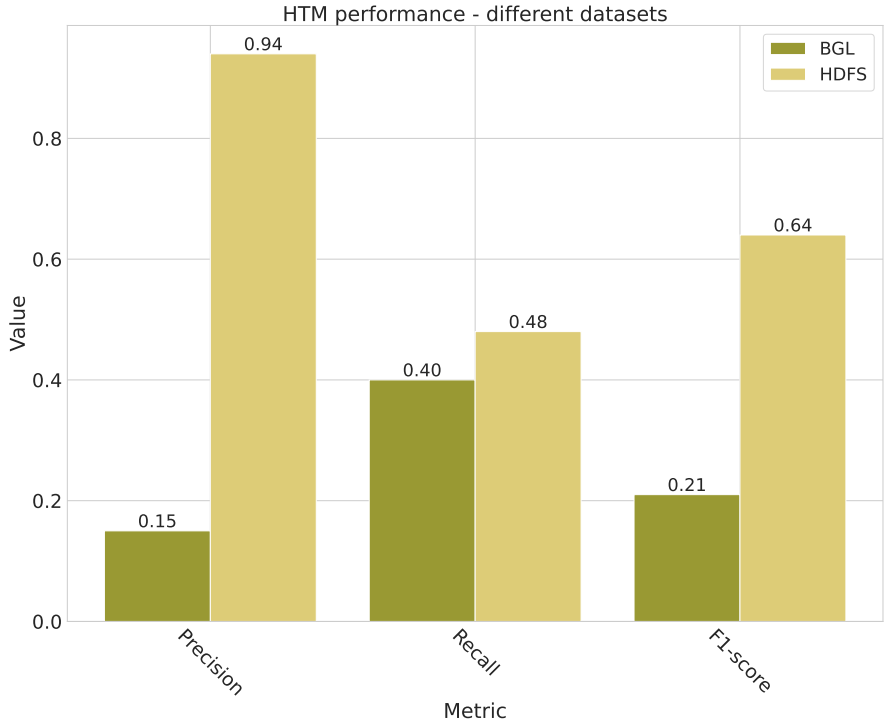

Figure 5.7: Performance of the model on the BGL and HDFS datasets.

The poor performance on the BGL dataset could be explained by the nature of the dataset. Since BGL contains no established log sequences or traces, the HTM struggles to learn sequences. With no fixed traces, logs appear randomly with no relation between two consecutive logs. This interferes with the sequence learning process of the HTM and leads to poorer results. However, there is still a possibility for an alternative explanation that the feature extraction techniques fail to capture representative features in the BGL dataset.

5.6.3 Ability to adapt to different tasks

These experiments investigate the model's ability to quickly adapt to different datasets or tasks and its ability to learn to predict anomalies in significantly different data.

Experimental setup: A custom dataset is created by combining logs of the HDFS dataset and the BGL datset. The first 3970 log sequences or blocks are from the HDFS dataset followed by 1414 log sequences from the BGL dataset and a final 3970 log sequences from the HDFS dataset. Figure 5.8 provides an illustration of the arrangement of the custom dataset. As earlier, features are extracted through sentence embeddings, reduced through PCA and encoded using scalar encoders.

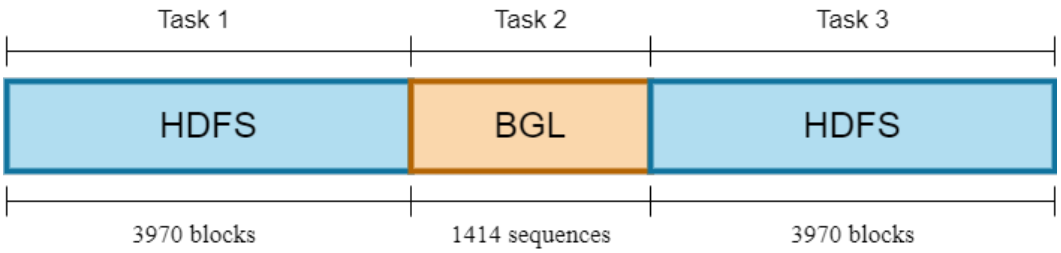

Figure 5.8: Visual representation of the arrangement of the custom dataset

By constructing such a dataset, the results of the experiment would allow for the following to be verifiable:

1. Does the model trained on task 2 perform better on task 3 indicating the possibility of transfer learning?
2. Does the model experience catastrophic forgetting from task 1 to task 3?

Sentence embeddings for the log messages of the custom dataset are extracted and reduced to 4 dimensions through PCA before using scalar encoders to encode the vectors as SDRS. The parameters of the encoders are the same as in Table 5.3.

Results: Figure 5.9 plots the average precision for the entire custom dataset. As expected, the model performs poorly during the BGL phase and performs much better for both the HDFS phases. It is also noticeable that the model quickly adapts to each dataset, reaching peak performance in around $20 - 30$ log sequences.

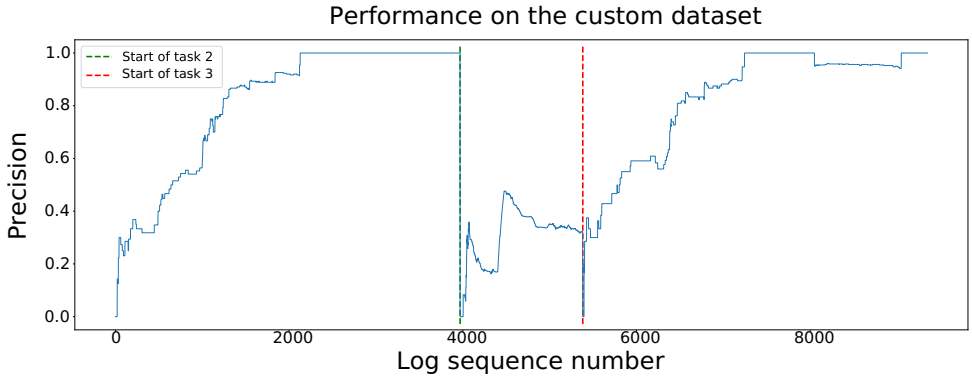

Figure 5.9: Average precision over time of the HTM model over the custom dataset.

In order to better understand if the model retains any information learned during the training of the first 3790 HDFS blocks, the performance of the model trained under three different circumstances is evaluated. The three different conditions are:

1. A model trained on the HDFS-BGL dataset.
2. A model solely trained on the full first 3790 HDFS blocks.
3. A previously untrained model.

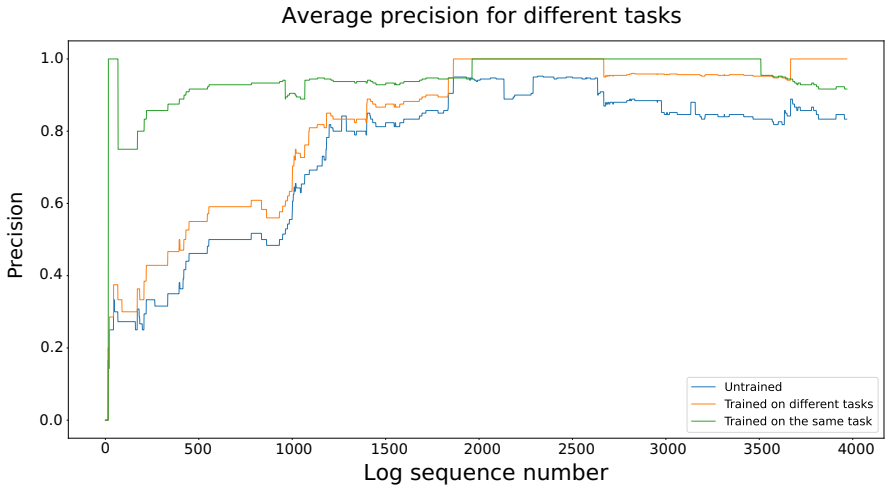

Figure 5.10: Comparison of average precision for HTM model trained in three different ways

The average precision is shown in Figure 5.10. The performance of the model trained on the HDFS model is as expected with high scores of precision. However, the model previously trained on the HDFS-BGL dataset and a completely untrained model display similar levels of precision throughout. This implies the following

1. While the model is able to adapt to new data very quickly, it does not possess any effective capabilities of transfer learning. This can be inferred from the untrained model performing similarly to the model trained on the HDFS-BGL dataset.

2. The model experiences catastrophic forgetting similar to artificial neural networks, it does not retain any information learned from training on the first 3790 HDFS blocks.

The above two conclusions answer the sub questions posed earlier in this subsection and with the results above it is safe to say that the HTM model is capable of quickly adapting to different tasks or datasets. However, experimental results suggest that the model does not retain any usable information from its earlier training and is incapable of transfer learning in this particular scenario studied in this thesis.

5.6.4 Online learning

It is vital for a log anomaly detection model to learn from new data in a continuous online manner as with every software update or patch, there is a chance of new log messages being introduced. In this section, the ability of the HTM to learn continously from streaming log data is evaluated by considering two models performing anomaly detection on the HDFS dataset with online learning disabled in one HTM model and with online learning enabled in the other HTM model. With online learning disabled, the HTM model does not grow new synapses or connections to other cells and does not update the permeance values of its existing connections. It

simply makes predictions to calculate the anomaly score.

Experimental setup: Two HTM models, one with online learning enabled and the other trained on 10% of the dataset while providing inference on the remaining 90% of the dataset are used to perform anomaly detection on the HDFS dataset. As earlier, the performance is evaluated by comparing the precision, recall and F_1 score of the models.

Results: The performance of the two models is displayed as a bar graph in Figure 5.11. As expected, the model with online learning enabled performs better than the one without online learning.

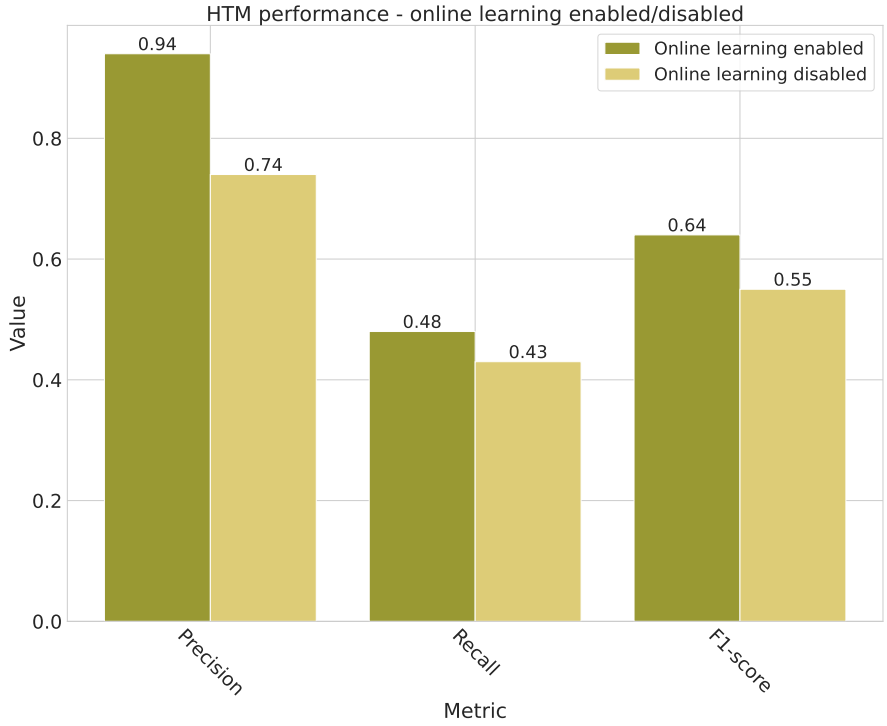

Figure 5.11: Performance of HTM model with online learning and with online learning disabled

To better understand the results, the average precision is calculated and plotted as well in the Figure 5.12

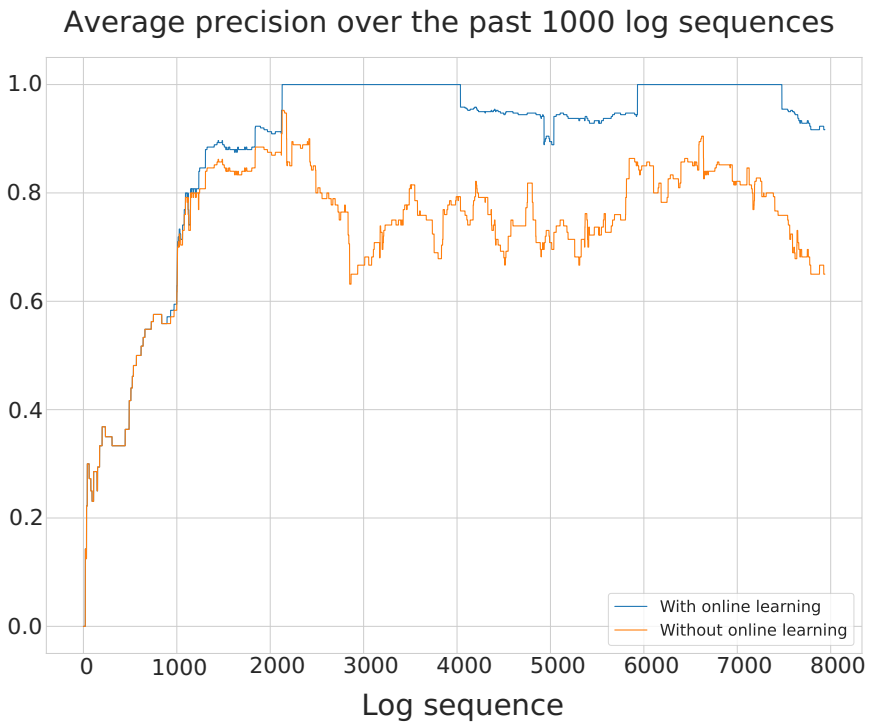

Figure 5.12: Precision over time for an HTM model with online learning enabled and with online learning disabled

From the figure, it is clear that after the initial 10% training data, the online model continuously outperforms the model with online learning disabled. Online learning helps the Spatial Pooler region of the HTM model to lean to map new unseen log messages as SDRs suitable for the Temporal Pooler while the Temporal Pooler learns new log sequences that may appear and even existing log messages that may appear in newer and different contexts.

5.6.5 Benchmark performance

To ensure that the HTM model is capable of perforning anomaly detection on real world data, it is essential to understand how well the HTM model fares against existing solutions. Here, the performance of several existing solutions are compared against the performance of the HTM model. In the case of the HDFS dataset, Loglizer[12] provides a framework that includes implementations of several popular log anomaly solutions. Table 5.4 displays the benchmark results of several supervised and a few unsupervised approaches to anomaly detection along with the results obtained from the HTM model.

Method	Precision	Recall	F_1 Score
Decision Tree	0.998	0.998	0.998
LR	0.955	0.911	0.933
SVM	0.959	0.970	0.965
One-Class SVM	0.995	0.222	0.363
Isolation Forest	0.830	0.776	0.802
HTM	0.943	0.484	0.640
PCA	0.490	0.610	0.582
IM	0.833	0.980	0.915
LogAnomaly	0.970	0.940	0.960
Deeplog	0.963	0.952	0.957

Table 5.4: Benchmark results of various existing models on the HDFS dataset.

Methods such as SVM [15] and Decision trees [4] are supervised approaches and since the HTM model is an unsupervised continously learning model, it is fair to compare the model with other existing unsupervised models. Benchmark results of other models are retrieved from their respective papers [22], [9], [17].

Figure 5.13 compares the results of the HTM model with other unsupervised models on the HDFS dataset. The bar graph clearly shows the superiority of deep learning models such as LogAnomaly and Deeplog which significantly outperform other existing models. However, despite being trained on a subset of the HDFS dataset, the precision of the HTM model at 0.94 is comparable to the LogAnomaly's and Deeplog's precision of 0.97 and 0.96 respectively. Furthermore, the HTM model outperforms the PCA based anomaly detection method across all three metrics.

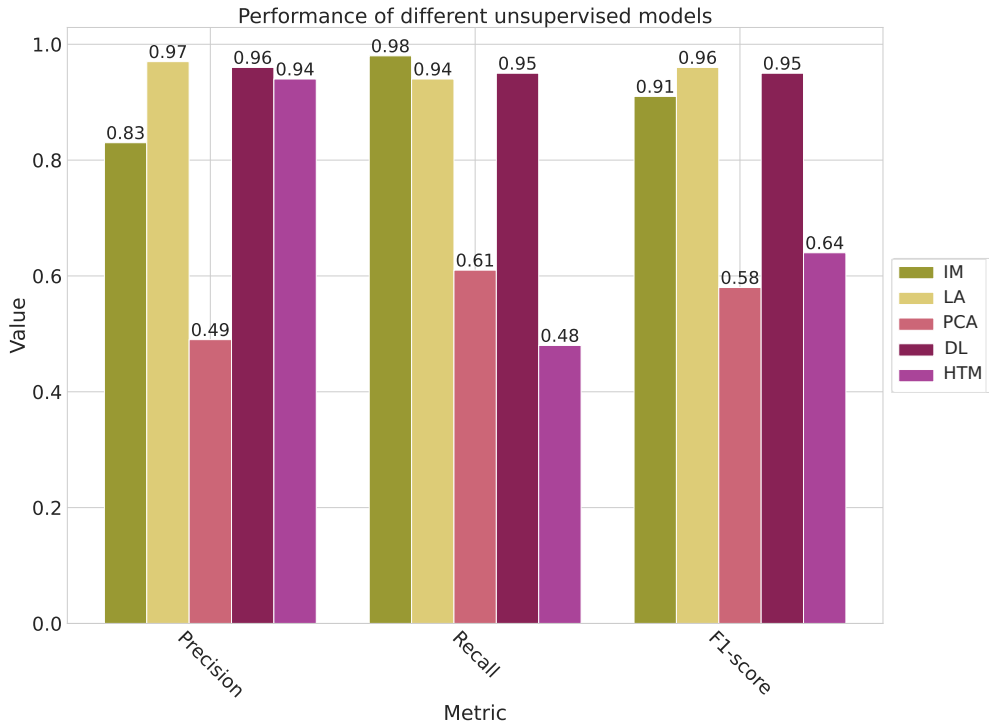

Figure 5.13: Comparison of the HTM model against existing unsupervised solutions. Note that Deeplog and LogAnomaly have been abbreviated to 'DL' and 'LA' respectively in the legend.

The results indicate that the HTM model performs better than earlier approaches to log anomaly detection such as PCA but lags behind current state of the art deep learning methods. However, it is also important to take into account HTM's online detection and learning ability. Deeplog is the only other model that claims to be capable of online learning and detection.

Impact on practical applications: The third research question evaluates a variety of different requirements that one might require from a log anomaly detection model from a business perspective. The HTM model is quick to adapt to different data and requires very little training data. Furthermore, the HTM has shown to be capable of continuous online learning and does not require constant periodic retraining that is often required for other online deep learning models. Furthermore, the HTM model is unsupervised and requires very little computational power compared to other models.

However, the HTM model requires that the data adhere to strict conditions in order to detect anomalies as shown by its poor performance on the BGL dataset. Repeating log data as well as undefined log sequences are major issues that the HTM model can not handle. Additionally, the HTM model requires that the data be converted to SDRs before being processed by the HTM model, whereas the existing

literature lacks an effective way of converting multivariate data with large number of dimensions into SDRs. These additional constraints might not be feasible in a real life scenario where the distribution of the log data as well as the stability of the data are things that can not be measured before hand. Lastly, the current implementation of the HTM model [6] is not very efficient either, as training for long periods of time reduces the processing speed of the model.

[6] https://numenta.com/htm-implementations/

Chapter 6

Conclusion

This section provides a thorough discussion on the results of the experiments, any major contributions of the thesis and provides an overview of the possible directions to take when considering future work.

6.1 Summary of the results

The thesis successfully implements an HTM solution to the classic log anomaly detection problem. The first research question is answered by investigating multiple feature extraction techniques. Experimental results indicate the superiority of sentence embeddings over other categorical approaches with the HTM model trained using sentence embeddings reaching F_1 scores of 0.64 on the HDFS dataset whereas the categorical approaches trail behind at 0.60 and 0.55. Using PCA to perform dimensionality reduction produces the best results for anomaly detection with precision, recall and F_1 scores of 0.94, 0.48 and 0.64, respectively, while usage of UMAP results in much poorer scores at 0.19, 0.37 and 0.25.

The experiment comparing four unique methods to encode SDRs provides answers for the second research question trying to find the most suitable approach to encoding log messages as SDRs. Scalar encoders have proven to be the most optimal with the highest F_1 at 0.64 and the second highest precision value at 0.94. The K-d tree approach of encoding messages manages to reach the highest levels of precision at 0.98 but consequently, its performance on recall is rather low at 0.18 rendering scalar encoders to be the most optimal. The performance of autoencoders and FHE are mediocre with F_1 scores of 0.50 and 0.45 respectively.

Finally, a series of tests conducted on the HTM's capabilities help answer the final research question dealing with the feasibility of the HTM model for log anomaly detection. Results from tests conducted on the HTM's online learning capabilities, transfer learning and adaptability to different tasks indicate that the HTM is capable of online learning resulting in an increase in performance as well as being adaptable to different tasks or changing datasets. The HTM model with online learning enabled produces precision, recall and F_1 scores of 0.94, 0.48 and 0.64, respectively, whereas the model with online learning disabled results in much lower scores at 0.74, 0.43 and

0.55. However, the performance does not translate to other benchmarking datasets such as the BGL dataset which lack log traces or sequences. The HTM model results in a much lower F_1 score of 0.21 for the BGL dataset.

In the context of application, the HTM model performs significantly better than other unsupervised approaches such as PCA but still lags behind current state-of-the-art models such as Deeplog. Additionally, the tedious amount of assumption on the data distribution that are required for the HTM model to perform well may hinder the implementation of the model for practical applications that deal with log messages. While the HTM has the advantage of being unsupervised and online, requiring little training data and almost no retraining, deep learning models such as Deeplog and LogAnomaly significantly outperform the HTM model and are capable of online learning. The major drawback of such deep learning models is their requirement for periodic retraining to ensure the model stays up to date in case the data distribution is non-stationary.

6.2 Summary of the main contributions

The thesis proposes a novel method to adapt the HTM model for the purpose of log message anomaly detection. The proposed methodology follows the different stages in log anomaly detection and provides contributions with strong affinity toward the HTM model in the following ways:

1. Firstly, two novel feature extraction methods, one utilizing sentence embeddings extracted using a transformer model and another using categorical templates weighed by IDF values are proposed and verified as suitable feature extraction techniques. The performance of the HTM on log anomaly detection shown in Figure 5.3 displays the method of sentence embeddings reaching an F_1 score of 0.64, recall of 0.48 and high precision of 0.94. The IDF - Categorical technique proves to be an improvement over the existing trivial categorical method with an F_1 score of 0.60 compared to 0.55, the F_1 score of the categorical approach. While both these approaches perform better than the trivial approach, the sentence embeddings technique with the highest F_1 score at 0.64 is the optimal method for log anomaly detection.

2. Secondly, the thesis proposes four unique approaches to encoding SDRs and evaluates their performance on log anomaly detection. The novel encoding methods include a K-d tree approach, autoencoders, the Fly-hash algorithm and a trivial scalar encoder.

   The four encoders result in the varying performance of the HTM model as evidenced by the results in Figure 5.5. The Scalar encoding performs the best out of the four with the highest F_1 score at 0.64 and is the most suited for log anomaly detection. However, the K-d Tree approach provides the highest precision score at 0.98 but also results in the lowest recall score at 0.18. The low F_1 scores of K-d tree at 0.31 and the remaining encoders of FHE and autoencoder at 0.50 and 0.45, respectively, render them unsuitable for log anomaly detection. However, owing to their ability to transform data to SDRs,

they might be interesting for future work regarding downstream tasks such as clustering or classification.

3. Finally, the work investigates the different capabilities of the HTM model and evaluates the model against existing state-of-the-art anomaly detection models. Several tests are performed on various features of the HTM ranging from its ability to perform continuous unsupervised online learning, transfer learning and its ability to adapt to different tasks to its processing speed. Furthermore, the thesis evaluates the viability of the HTM model in a real-life business situation by comparing its performance across datasets and under various scenarios that the model might encounter in a production environment.

   Figure 5.7 shows the HTM performing rather poorly with the BGL dataset with an F_1 score of just 0.21 implying that the HTM model cannot directly perform log anomaly detection on all datasets. Additionally, the processing speed of the HTM gradually decreases with increasing data as seen in Figure 5.6. With these drawbacks in mind and comparing the performance of the HTM model to SOTA in Table 5.4, the current HTM model is not the best suited for log anomaly detection in a production environment. However, it is important to note that the HTM is the only model capable of continuous online learning and, as evidenced in Figure 5.11, the HTM model's performance drastically improves with online learning enabled.

In short, the work provides two different feature extraction techniques, four unique ways to encode SDRs and a complete and thorough investigation into the capabilities of the HTM model for the task of interest: Online Anomaly Detection on streamed log messages.

6.3 Future work

There are three possible primary areas where future work could aid in the development and improvement of the proposed solution to the anomaly detection problem. The first is to focus on feature extraction part of analyzing log data. Currently, information is abstracted through the use of log templates solely. Information on process Ids, log parameters such as instance Ids and machine Ids are simply abstracted. Instead, through an appropriate method of feature encoding, these additional features might help improve the performance of the HTM model. Additionally, improvements to the existing sentence embeddings could be made through training a transformer model on a custom corpus relating to the log dataset.

The second area is the representation of the data as SDRs including investigating alternative methods for encoding the data as SDRs. At present, the four different approaches that are explored in the thesis are general purpose encoders designed to be able to encode any scalar or categorical data. An encoding approach that is primarily focused on encoding word or sentence embeddings might provide a more effective representational SDR for the spatial pooler of the HTM model. A starting point would be to implement an encoder following the semantic folding technique described in Cortical.Io's white paper [29] as it could provide greater degree of accuracy in

preserving distances globally and locally in the word embedding space.

The final area of interest is the HTM model itself where improvements could be made to the implementation of the model to better adapt the model to machine learning problems. The current implementation of the model strictly assumes temporal distribution of the data in addition to assuming the data to be in the form of fixed sequences. Additionally, the HTM model is unable to handle more than four or five features due to the size of the SDRs. Therefore, using an ensemble model with a combination of a traditional Long Short Term Memory (LSTM) and an HTM model might help alleviate some of the weaknesses of the HTM model.

Appendix A

Parameters of the HTM model

Parameters	**Value**
boostStrength	2.0
columnCount	(2048)
localAreaDensity	0.04
potentialPct	0.85
potentialRadius	2363
synPermActiveInc	0.04
synPermConnected	0.139999999999
synPermInactiveDec	0.006
activationThreshold	17
cellsPerColumn	16
initialPerm	0.21
maxSegmentsPerCell	128
maxSynapsesPerSegment	64
minThreshold	10
newSynapseCount	32
permanenceDec	$0.1
permanenceInc	0.1

Table A.1: Parameter values of the HTM model used in the experiments.

Bibliography

[1] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017, online Real-Time Learning Strategies for Data Streams. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217309864

[2] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, ser. Proceedings of Machine Learning Research, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., vol. 27. Bellevue, Washington, USA: PMLR, 02 Jul 2012, pp. 37–49. [Online]. Available: https://proceedings.mlr.press/v27/baldi12a.html

[3] W. Cao, X. Feng, B. Liang, T. Zhang, Y. Gao, Y. Zhang, and F. Li, "Logstore: A cloud-native and multi-tenant log database," 06 2021, pp. 2464–2476.

[4] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, 2004, pp. 36–43.

[5] Y. Cui, S. Ahmad, and J. Hawkins, "The htm spatial pooler – a neocortical algorithm for online sparse distributed coding," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/02/16/085035

[6] S. Dasgupta, C. F. Stevens, and S. Navlakha, "A neural algorithm for a fundamental computing problem," *Science*, vol. 358, no. 6364, pp. 793–796, 2017. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aam9868

[7] J. Deng, Z. Zhang, E. Marchi, and B. Schuller, "Sparse autoencoder-based feature transfer learning for speech emotion recognition," in *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, 2013, pp. 511–516.

[8] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, 2019.

[9] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," 10 2017, pp. 1285–1298.

[10] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[11] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, "Biological and machine intelligence (bami)," 2016, initial online release 0.4. [Online]. Available: https://numenta.com/resources/biological-and-machine-intelligence/

[12] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. IEEE Computer Society, 2016, pp. 207–218. [Online]. Available: https://doi.org/10.1109/ISSRE.2016.21

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[14] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1356–1367. [Online]. Available: https://doi.org/10.1145/3510003.3510155

[15] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," 11 2007, pp. 583 – 588.

[16] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 102–111. [Online]. Available: https://doi.org/10.1145/2889160.2889232

[17] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," 01 2010.

[18] S. Lu, X. Wei, Y. Li, and L. Wang, "Detecting anomaly in big data system logs using convolutional neural network," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 151–158.

[19] A. Makhzani and B. Frey, "k-sparse autoencoders," 12 2013.

[20] M. Manassi, B. Sayim, and M. H. Herzog, "When crowding of crowding leads to uncrowding," *Journal of Vision*, vol. 13, no. 13, pp. 10–10, 11 2013. [Online]. Available: https://doi.org/10.1167/13.13.10

[21] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018. [Online]. Available: https://arxiv.org/abs/1802.03426

[22] 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 *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 4739–4745. [Online]. Available: https://doi.org/10.24963/ijcai.2019/658

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[24] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 575–584.

[25] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," Nov. 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[26] S. Purdy, "Encoding data for htm systems," 2016. [Online]. Available: https://arxiv.org/abs/1602.05925

[27] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[28] C. Tao, H. Pan, Y. Li, and Z. Zou, "Unsupervised spectral–spatial feature learning with stacked sparse autoencoder for hyperspectral imagery classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 12, pp. 2438–2442, 2015.

[29] F. D. S. Webber, "Semantic folding theory-white paper," 2015.

[30] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132. [Online]. Available: https://doi.org/10.1145/1629575.1629587

[31] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1448–1460.

[32] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on*

European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817. [Online]. Available: https://doi.org/10.1145/3338906.3338931

List of Figures

List of Tables