INF FAKULTÄT FÜR INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department of Technical and Business Information Systems
Very Large Business Applications Lab

Log-Based Anomaly Detection In Open Source Cloud Infrastructure Automation Solution Using Machine Learning

Master Thesis

Author:
Deeksha Ramakrishna

Examiner and Supervisor:
Prof. Dr. Klaus Turowski

2nd Examiner:
Dr.-Ing. Eike Schallehn

Supervisor:
MSc. Andrey Kharitonov

Magdeburg, 12.06.2023

Deeksha Ramakrishna
Log-Based Anomaly Detection In Open Source Cloud Infrastructure Automation Solution Using Machine Learning
Master Thesis, Otto-von-Guericke-University Magdeburg, 2023.

Abstract

Cloud is being used pretty much everywhere nowadays and cloud service providers have to keep it up and running for thousands of customers who run their services using cloud. As it scales up rapidly, it is very much critical and required to monitor this continuously to detect the failures that could affect the customers using it. SysEleven Stack is based on the Openstack which provides services such as compute, storage, network, identity service and so on. All these services are being used by 500 plus customers across European Union.

The cloud this size would produce huge number of log entries through different services. These log entries contains very important information with regards to the behaviour about the cloud services. Because of its huge number of continuous entries it is extremely difficult to get the desired information required for troubleshooting purposes, manual analysis for failures, anomaly detection and so on.

Existing approach in the SysEleven stack is to define a threshold value by defining alerting rules to detect the failures or abnormality in the infrastructure. Defining such rules is a tedious task and it does not notify upcoming problem to the Site Reliability Engineer(SRE) until and unless the threshold is reached. This is a very expensive model. So there is no way to know or to detect if there is any behavioral changes that could give heads up to fix problematic incidents and avoid a future production downtime.

Various Machine Learning and Deep Learning algorithm are used for the anomaly detection. Several algorithms of supervised and unsupervised learning are used to detect anomalies by employing the log parsing and dimensionality reduction techniques.

In this thesis the efforts are being made to build an end to end system of detecting the anomalies using the machine learning algorithms without having to define the threshold value for every anomalies that occur. The thesis also describes the comparative study of the algorithms with respect to anomaly detection in the OpenStack logs data.

The solution would help the SREs to find the anomalies and to monitor the infrastructure for faulty behaviours in order to quickly debug and fix it in a timely manner.

Acknowledgements

I would like to thank the Department of Very Large Business Applications(VLBA), Otto-von-Guericke University, Magdeburg for accommodating me to do my thesis. I also would like to express my gratitude to Prof.Dr. Klaus Turowski for teaching about the Cloud and allowing me to pursue my thesis under his chair.

I would like to express my special appreciation and thanks to my supervisor Mr. Andrey Kharitonov for his constant support and guidance during this research.

I would like to extend my gratitude to Mr. Jens Plogsties for giving me an opportunity to conduct this research in SysEleven and my second supervisor Mr. Bernd May for his guidance. I am sincerely grateful to the CloudStackers team for their support and contribution in knowledge transfers.

Last but not least, I would like to thank my family and friends for their constant support and kind words during this journey.

Statement of Authorship

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Signature

Place, Date

Contents

List of Figures

1 Introduction

The end users of the cloud services would expect it to be highly available and reliable in order for them to run their applications. The cloud is being treated as a product that is delivered to the customers and its growth is achieved based on how secure, reliable and available the platform for the end users is. Hence huge investments are made by the cloud service providers to keep the cloud up and running for the customers.

It provides different infrastructure services such as compute, storage, network, identity and so on required by the organizations or an individuals to configure and run their applications or desired use cases.[Opeb]

Several research projects have been carried around the cloud to keep it available 24/7 to all the end users and one such work is detection of anomalies in the cloud services. The cloud is a big set up which produces huge number of logs everyday and these logs consists of important information about the behavior of the cloud. Monitoring these behaviors helps the experts to keep the cloud healthy.

The log messages indicate the system state which gives insights about the possible failures or behavioral changes that needs to be addressed. They usually start with the timestamp and followed by the message. Monitoring the cloud is a necessity as they record all the errors, warnings, information that arrive from the embedded software which runs on the nodes. These are the crucial information which consists of the status regarding the cloud health. Hence SREs put efforts in to keeping track of the cloud behaviors available for its customers by configuring the monitoring tools like Prometheus[1], Grafana Alerting[2] which requires predefined rules with thresholds.

There are several researches and implementations already been done on the detecting the anomalies in the cloud using the traditional algorithms of machine learning, deep learning, using the log parsing algorithms, but there is still room for improvement in terms of performance and efficiency and hence defining the new methodologies in resolving the reliable automatic monitoring system of cloud could reduce the human efforts in detecting them.

[1] Prometheus[Pro]: It is a open source tool used for monitoring and alerting which requires defining of the alerting rules and these alerts are sent to appropriate receiver such as Email, Opsgenie via the alertmanager

[2] Grafana Alerting[Gra]: It is a open source software that allows to learn about the problem in the system in a consolidated view through dashboards and take the necessary actions.

1.1 Motivation

At SysEleven[Sys], the cloud services are deployed using the OpenStack. Monitoring the failures, abnormalities are very much expected when using such services and making the cloud services available for the users is very vital since their applications and services are deployed using it. Hence, in case of failures, diagnosis and fix would be very time dependent and challenging in order to maintain the robustness of the cloud.

Various distributed service daemons of the OpenStack components such as nova, cinder, keystone, glance and so on generate relevant logs which will record the traces of its execution in the respective log files. Current system involves defining the rules for the expected and known issues using queries and defining the threshold values. There is no way to recognize the failures or abnormalities automatically until the threshold is reached. Once the system reaches the threshold an alert is triggered only for those pre defined rules. Unknown failures might sometimes be notified by customers when they are unable to reach certain services or some services does not behave as expected or it can only be observed when an SRE notices something unusual.

Hence, until an external support ticket or internal observation occurs, there is no information about the change in the usual trend. By the time the cloud support gets the ticket and act upon it, a substantial time would have been passed. These problems could bring the entire cloud down when left without a fix and could cause the production downtime for the customers and in turn affect the business.

Therefore there exists a need for an automatic detection of anomalies for keeping the cloud robust and to take actions whenever necessary in a timely manner.

1.2 Main contributions

As per the above motivation this thesis concentrates on addressing the requirement of automatic anomaly detection in a more reliable and efficient way in SysEleven by using the Machine Learning based approaches.

1. In order to build an automatic detection of anomaly, we would be exploring the state of the art and novel algorithms to address the anomaly detection. Hence a comparative study about different types of state of the art algorithms to detect the anomalies are being implemented.

2. We evaluate the performance of the novel and state of the art algorithms by using OpenStack logs as the dataset. The algorithms Copula based outlier detection and Unsupervised outlier detection using Empirical distribution Functions are used for the first time for the log based anomaly detection in cloud and OpenStack to the best of our knowledge. As they claim to perform well in case of outlier detection, it would be a good fit to test them.

3. The performance of the ensemble models are evaluated for the detection of anomalies.

4. Experimental field study by deploying the end to end machine learning model along with the alerting system to the dedicated anomaly detection server in the overlay infrastructure in SysEleven stack.[3]

1.3 Research Questions

The below mentioner research questions are addressed in this thesis.

1. [**RQ1:**] **Are the chosen machine learning algorithm are able to accomplish the business use case?** Different kinds of machine learning algorithms are available which uses Natural Language Processing(NLP) techniques to detect the anomalies. This research question is subdivided in to the following.

   a) [**SRQ1.1:**] **Which state of the art or novel algorithm is considered as the best fit in comparison to the other algorithm of choice?**This thesis is concentrating on considering the 6 algorithms based on the literature survey and the state of the art, novel algorithms that fits for anomaly detection.

   b) [**SRQ1.2:**] **How can the models be evaluated and what are the qualitative and quantitative assessment that can be considered?** The thesis aims at using both the qualitative evaluation metrics such as recall, f1-score, precision and so on along with the domain expert feedback to evaluate the models.

2. [**RQ2:**] **How the ensemble approach behaves on the detection of anomalies and which algorithms contribute towards the better performance?**

   a) [**SRQ2.1:**] **Which is the best ensemble model that would fit the solution?** The models are used in as a combination and the best combination that perform better compared to the other combination of models and individual model's performance will be determined.

   b) [**SRQ2.2:**] **How can the ensemble models be evaluated?**The focus is on the evaluation approaches for the ensemble model to determine the best fit for the research problem.

[3]SysEleven Stack[Sys]: It is an OpenStack public cloud service from a Berlin based company known as SysEleven GmBH

1.4 Thesis structure

In addition to the above explanation, the rest of the thesis is structured as follows:

- In chapter 2 the Design Science Research methodology that has been used in the thesis to define a systematic way of solving the defined problem is explained. Each phase involved in the DSR is explained in detail in this chapter by aligning it with the anomaly detection.
- Next it is followed by the background study. The chapter 3 describes the background required to understand the use case in detail. The overview of the OpenStack, its deployment architecture, its logical architecture are explained and the existing detection of anomaly system being used in the business in explained.
- The chapter 4 explains the systematic literature review done on the use case which gives the overview of the prior research that has been done using the Machine Learning in case of the anomaly detection in cloud, log based anomaly detection in cloud and also detection of anomalies in the OpenStack logs.
- The chapter 5 explains about the theoretical knowledge that is required to implement the solution. This involves the description of the preprocessing techniques, feature engineering, models, evaluation metrics.
- This is followed by the explanation of conceptual design of the solution in the thesis which gives the overview of each step in the implementation of the solution. This is explained in chapter 6.
- The chapter 7 consists of the implementation details about the execution of experiments using the selected models for analysis.
- The chapter 8 gives the explanation about the results and evaluation to evaluate each of the model performance and to select the best fit model for the problem defined.
- The chapter 9 explains the details about the conclusion that has been drawn from all the experiments in the thesis and also it describes the answers to all the research questions defined in this study, the limitations and challenges faced in the study are described and also the potential future works and ideas that can be carried out for this problem to find the better solutions are discussed briefly.

2 Methodology

Anomaly detection is the actively researched area. Several practical application uses anomaly detection to detect the change in trend, unusual incidents and so on that would bring down the entire system down or could create a detrimental effect on the business such as failing to meet the Service Level Agreement(SLAs) if gone unnoticed or ignored. In cloud it is very crucial that these anomalies must be addressed else customers using the cloud services would be affected and in turn would cost the cloud service providers.

Using machine learning to detect the anomalies in general and with respect to cloud has been actively researched. Though there are solutions exists to tackle this learning problem, there is still room for improving the performance and to use state of the art algorithms to make the model more efficient. The focus of this thesis is to develop a research prototype that would allow us to answer the defined research questions. Hence this prototype is done systematically by following the Design Science Research(DSR).[Hev07]

DSR aligns with the system development methodology in order to go through the process that allows to get the artefact required. When doing the DSR there are 2 concepts that are of at most importance i.e, creation of something and the process or method followed to create the process. Design science is considered as an explicitly organized rational and systematic approach to the design[Hev07]. In the DSR essay Hevner focused on the practical nature of DSR when he referred to the "fundamentally a problem solving paradigm". The nature of DSR can be "Pragmatism as Paradigm" meaning create something that works in practice.[Hev07]

The reason to do DSR is to get the artefact needed by performing a systematic process. This artefact could be "Methods", "Implementation" or a "Design Theory". The explanation about this artefact are given in the table below.

Table 2.1: Description of Artefact

Artefacts	**Description**
Methods	It is a procedure for achieving something in a systematic way
Implementations	The execution of a decision or a plan
Design Theory	It is a perspective that provides details that can help the practitioners and scholars to design an effective system

In this thesis, these artefact are achieved using a DSR approach. In figure Figure 2.1 the steps and the process of DSR is given. The problem needs to be obtained from environment which involves people, organization, technology. "Relevance" comes from the environment that is being used to get the problem in order to define the business needs.

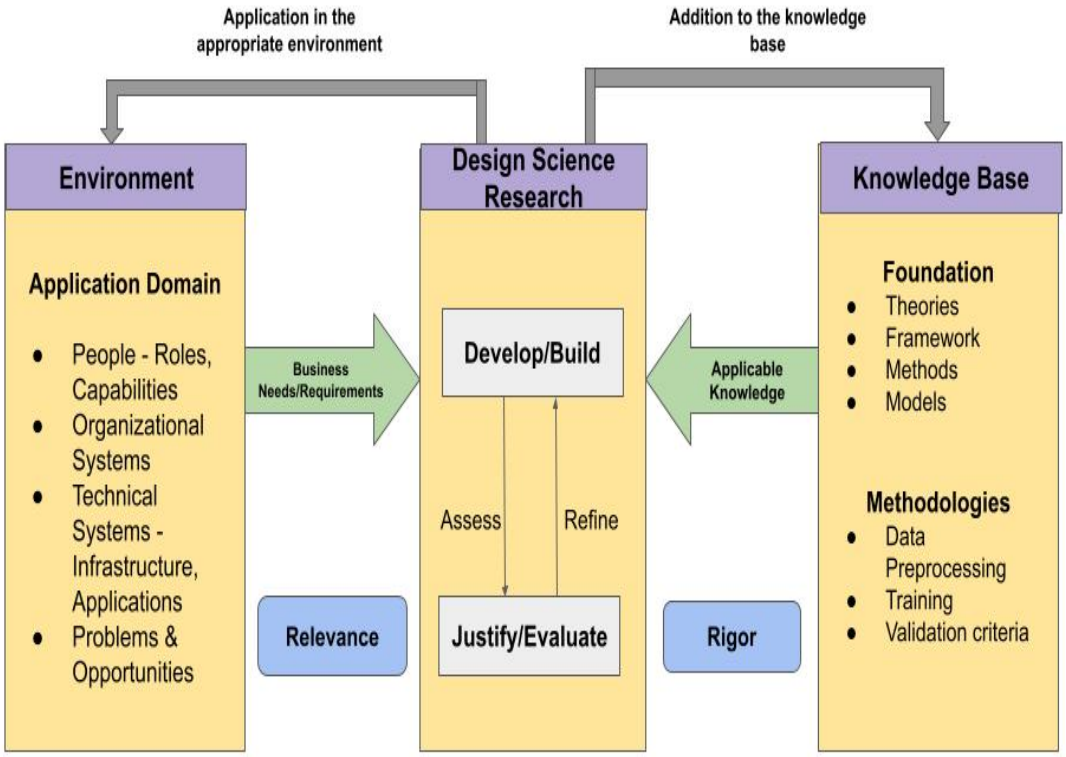


Figure 2.1: Design Sciences Research Cycle[Hev07]

The theory and literature is obtained from the knowledge base which involves the existing foundation, theories, framework, methods, models and also the existing methodologies which contribute to the process in order to build an artefact. The process that is followed to do DSR is shown in the middle section of the Figure 2.1. After developing the process to tackle the problem, the results must be evaluated rigorously using well-executed evaluation methods and based on the evaluation results, if it needs improvement it is again sent back for developing. A continuous process of assess and refine are carried out here. After developing the plausible solution to the problem, it is given back to the environment and the knowledge acquired from it back to the knowledge base. The arrows indicate that this is a cyclic process between the Environment, DSR and Knowledge base.[Hev07]

The primary focus in the first phase of DSR involves understanding the business needs and requirements to fulfill those needs that has been received in the form of a problem from the environment. This is very crucial as the rest of the process depends on the understanding of the objective of the problem. After understanding the business needs, a detailed research of the required foundation and the methodologies that can be performed such as data preprocessing involving data cleaning, transformation and integration, training and validation of the results must be done to build the solution. The last phase focuses on building actual solution which involves building the model, hyperparameter tuning, training and evaluating the results in accordance with the business use case and background knowledge.

In this thesis, evaluation is carried out both by the quantitative evaluation metrics and the qualitative assessment by the domain experts. The last step is most critical as the solution defined must satisfy the business needs. DSR requires different evaluation criteria depending upon the context of the problem being addressed. Such evaluations will validate the research and findings. A consolidated purpose and evaluation activities must be designed to full fill the DSR process. The below table gives a summary of the evaluation activities that are opted and also includes the information about the criteria. The Ex post evaluation is applied to the implemented system in this thesis. [SvB12].

Table 2.2: DSR evaluation activities and criteria[SvB12]

Activity	**Input**	**Output**	**Eval.Criteria**	**Eval.Methods**
Qualitative evaluation of the results	Problem statement/ Observation of a problem, Research need, Design objectives, Design theory, Existing solution to a practical problem	Justified Problem statement, Justified research gap, Justified design objectives [SvB12]	Applicability, suitability, reliability, correctness, effectiveness, robustness, suitability	Literature review, Expert Opinion
Quantitative Evaluation	Design specification, Design objectives, Design tool	Validated design specification, Justified design tool [SvB12]	Feasibility, clarity, level of detail, applicability, robustness	Evaluation metrics, logical reasoning, demonstration
Cost effectiveness and Performance	Prototype	Validate model in an artificial setting	Feasibility, ease of use, efficiency, fidelity with real world phenomenon, operationality	Demonstration with experiment of a model, experiment with system

The detail explanation of the table Table 2.2 is explained below.

Qualitative Evaluation:

The quality of the model can be determined by its ability to detect the right anomalies based on interviewing the expert and their feedback on it.

- Input here can be a defined problem statement with respect to the task that needs to be accomplished. Requirement of the research that is needed to address the problem. The basic requirements to accomplish the task and possible outcomes can be used as design objectives. Design theory could be the systematic and structural approach followed to achieve the effective system for the use case defined. Existing solutions could be already used and tested methods to the tackle the problem.

- Output can be a problem statement that is well understood and satisfy the business needs, full filling the research gap in the existing research and approaches for the defined problem and it could also be the possible outcomes that were achieved based on the design objectives.
- Evaluation criteria in this can be measured based on the applicability meaning if the chosen idea or solution is suitable for the use case and business needs, suitability is whether the chosen concept, or solution have the required features or characteristics and capabilities to effectively satisfy the problem statement and if it aligns with the available resources, reliability is whether the chosen solution be used by others and can obtain the similar results, correctness is whether the result is plausible enough to accept it as accurate results, effectiveness which is the extent to which the proposed solution is successful in accomplishing the goal, robustness which involves the solution's capability to handle the variation and suitability tells us about the solution to fit the defined problem statement.
- Evaluation methods used to evaluate the qualitative result could be by literature review and also by expert opinion.

Quantitative Evaluation:

The quantitative evaluation involves the evaluation metrics like precision, recall, f1-score and so on to evaluate the model performance.

- Input can be design specifications which involves quality of data being passed, pre-processing, hyper-parameter tuning. Design objective is the basic requirements and possible outcomes as mentioned above. Design tools are the software applications, platforms or frameworks which can be used to develop the solution for the business problem.
- Output would be the specifications that are validated by performing the tests. Justified design tools that satisfy the needs of the solution development.
- Evaluation criteria could be feasibility meaning the solution proposed is practical and achievable, level of detail could be the detailed assessment of efficiency, performance, reliability, usability of the proposed system, applicability that determines with values in the result whether the solution could be applicable to the current needs and also robustness.
- Evaluation methods for this could include evaluation metrics, logical reasoning behind the model performance and demonstration that involves providing the evidence and how the model is performing on the real world scenarios.

Cost effectiveness and Performance:

This mainly deals with the compute power necessary to run the designed solution in the real world.

- Input involves the prototype of the solution to the problem.

- Output could be validating the solution by creating an artificial setting or development environment which determines the performance.
- Evaluation criteria involves feasibility of the system that has been tested, ease of using the solution, efficiency refers to the usage of resources, time, cost to accomplish the business needs, fidelity involves capturing the degree to which the proposed solution resembles the real world application, operationality refers to the practical and functional ability of a solution for the real world setting.
- Evaluation methods involves demonstration of the solution and experiment by deploying the solution to the test system.

The DSR aims at providing the systematic development to meet the business objectives and goals. The table below gives the detailed description on the phases of the DSR according to its guidelines and the reference on how they are incorporated in this thesis.

Table 2.3: Description of DSR phases in the thesis

Phases	Description	Reference
Environment	In this case the environment involves SysEleven Stack based on OpenStack which is used by 1000 plus customers across European Union, SREs who are involved in developing the cloud and maintaining it.	Introduction Chapter 1
Business Objective	To build an automated anomaly detection system to eliminate the human efforts to detect the anomalies manually by defining the rules.	Introduction Chapter 1
Research Focus	To demonstrate a Comparative Study of different machine learning algorithms on the basis of its performance and efficiency in order to determine the right model for the defined problem and also to evaluate the performance of the ensemble approach.	Research question Section 1.3
Knowledge Base	This phase focuses on the background knowledge of OpenStack, theories of the data preprocessing, machine learning model, training, evaluation criteria, existing approaches for the anomaly detection and so on required to achieve the goal.	Background and literature survey Chapter 3
Developing	This involves implementing the data collection, data preprocessing, Log parsing, model training and prediction.	Implementation Chapter 7
Evaluation	Both the qualitative and quantitative evaluation is done on the developed models to determine the best fit for the chosen problem.	Evaluation Chapter 8
Deployment	An existing deployment approach is adapted in this thesis and shown how can we deploy machine learning model in the sever placed in SysEleven overlay infrastructure.	Implementation Chapter 7

3 Background

This chapter explains about the background knowledge required in order to understand the need of business problem and the details of the environment the business use case is being used.

According to DSR[Hev07] understanding the business problem is a very important step in building the solution to the problem. Hence, this chapter briefly explains about OpenStack and it also explains the existing system to detect the problems in the SysEleven stack.

Studies have done on the OpenStack data with the algorithms like k-means, SVM(Support Vector Machines), LSTM, GMM and so on with PCA(Principal Component Analysis), TF-IDF, Word2vector for feature engineering. This thesis concentrates on the comparative study of the most used and state of the art models, novel models for the detection of anomalies in the OpenStack log data.

3.1 Anomalies

Anomaly is a condition of unusual or abnormal. It is an abnormal behavior that does not gel up with the rest of the patterns. The word anomaly is derived from the Greek word "Anomalos" which means irregular or uneven.

3.2 Logs

In computing, system logs are referred to as documentation of the events that occur in the computer system. They are in unstructured textual format that hold the information of timestamp, process ID and message that are printed directly by the source code which includes the system performance, user activities and application information.

```
"2023-03-01 10:17:59.561 43213 INFO nova.metadata.wsgi.server [req-3fce8a2f-21f9-487e↩
    -8045-c6f9d5cedd32 - - - - -] 10.0.0.115,10.33.48.47,10.33.48.27,127.0.0.1 ""GET /↩
    latest/meta-data HTTP/1.1"" status: 200 len: 347 time: 0.3453352"
```

Listing 3.1: Sample Nova Log

The above Listing 3.1 shows a sample of the OpenStack nova logs which are recorded by the nova-api.service and stored in the path defined in the service file. The logs consists of a

timestamp, followed by the PID, Verbosity level(INFO, DEBUG and so on) and the message related to the service. Using these logs the SRE can determine the system status and diagnose the root cause or fix the system when failure occurs.

3.3 OpenStack

"OpenStack is an open source distributed cloud operating system software written in Python that controls large pool of compute, storage, network resources through the datacenter"[Opeb]. Several organizations use OpenStack to develop the cloud based on the OpenStack software and they are all part of the development of OpenStack. The development can be managed and provisioned through the common authentication mechanism. It has different components which includes 5 core components namely Nova for compute services, Cinder for storage service, Keystone for identity management, Glance for image service and Neutron for networking. The projects are integrated for example Nova uses Neutron for provisioning the switches.[Opeb][Opec]

In the case of virtualization, resources such as storage, CPU, and RAM are gathered from different vendor-specific programs and are split by a hypervisor as required. OpenStack uses an application programming interfaces (APIs) to extract virtual resources and to put them in to a discrete pools. The basic idea is that the physical resource in datacenter(DC) are pooled and placed under the OpenStack software control.[Opeb][Opec]

There are 4 important nodes namely Cloud Controller node, Compute node, Storage node and Network node.

3.4 OpenStack Deployment Architecture

The Cloud Controller node(CCN) running all the control plane services which acts as a central management is deployed in a redundant configuration and it will have high availability. Hence if one node fails, another can take over the required control plane task. The CCN needs at least two network interfaces, one connected to the external network so that the API clients and end users can access the services and make API requests to the OpenStack service endpoints. The other interface connected to the management network to enable control plane communications with other nodes in the deployment such as sending Advanced Message Queuing Protocol (AMQP) messages, Remote Procedure Call(RPC), downloading images, monitoring, management and so on. Services such as are SQL database, Message queue(Rabbit MQ), Network time service(for time synchronization), Keystone(Identity service), Glance(Image service), User dashboard(Horizon), Nova API, Nova Conductor, Nova Scheduler, Nova Metadata Agent, Neutron server, Cinder API, Cinder Scheduler, Object Storage Proxy service, Telemetry management service and Heat Orchestration service runs on the CCN.[Opea]

The network node that hosts the router and Dynamic Host Configuration Protocol(DHCP) name spaces, provides data plane communications between tenant networks via the data network and also routes traffic to and from the external network. This node has high availability as it handles all the up-link traffic from tenant networks and it works as a channel between the tenants network and the external networks in the DC and the link

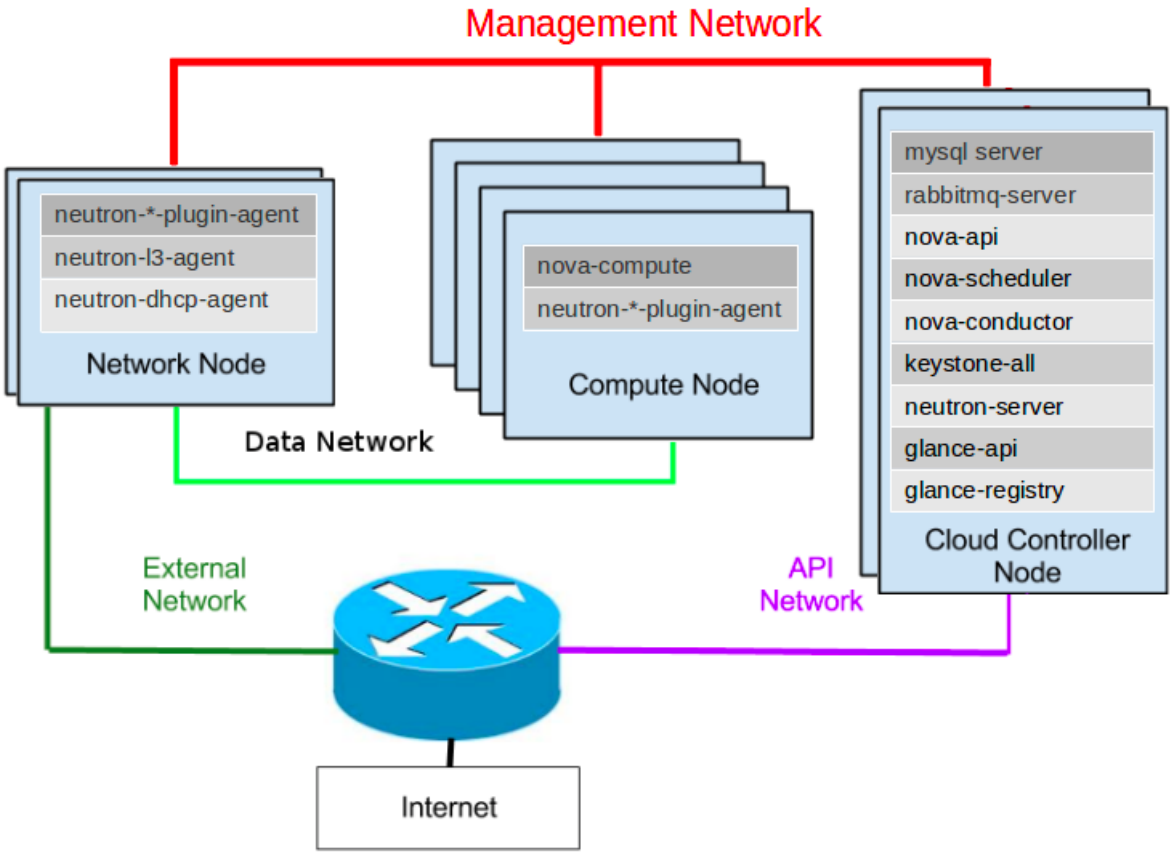


Figure 3.1: OpenStack Deployment Architecture[Opea]

bonding techniques such as LACP(Link aggregation control protocol) is used to provide link redundancy and network throughput. This process is carried out by the neutron-server which is a daemon processs. This is a critical node and hence if this goes down, all external and internal network connectivity will be broken.

The network node has 4 network interfaces namely,

- Management network: Used for controlling communication through controller.[Oped]
- External network: Used to route the traffic to the external network meaning the network that exists outside of OpenStack.[Oped]
- Data network: Used by the VMs to communicate. [Oped]
- API network: It provides all OpenStack APIs to the tenants. Any IP addresses that are in the API network range can be accessed by anybody on the internet.[Oped]

The compute nodes are used as the resource core of the OpenStack cloud running the hypervisor program that virtualizes the hardware platform and provides the necessary CPU, memory, network and storage resources to run the various tenant instances. These nodes are horizontally scaled in any deployment meaning, to scale the cloud and to increase the compute capacity, more of these nodes must be added. The default hypervisor of nova is Kernel Based Virtual Machine(KVM) and the tenant instances are schedules on this node by nova scheduler program. The compute node runs the nova compute process and

this is responsible for creating and managing Virtual Machines(VMs) by talking to the hypervisor by libvirt driver.

Compute node has 3 network interfaces. Data network is used for data plane communications i.e, applications running inside the tenant instances when they communicate with one another and to the outside world, that traffic goes over the data network and the interface attached for the management network is for control plane communications. For example the nova scheduler on the controller node messaging the nova compute process running on the compute node to create a VM happens via the management network. Storage network is used for mounting the storage volumes provisioned by cinder onto instances.[Opea]

Storage node contains the disks used for providing the persistent storage volumes to project instances. In case of Logical Volume Management(LVM) storage backend, several physical volumes comprising of one or more physical hard drives are combined into one logical volume service to create persistent volumes for instances. The storage volumes are allocated by cinder from a storage backend such as Linux LVM or Ceph RBD(RADOS block device) or any other storage devices from various vendors and they are integrated into cinder by using volume drivers. By default Cinder make use of LVM backend. Storage nodes have 2 network interfaces which is attached to the storage network and management network.[Opea]

3.5 OpenStack Logical Architecture

End users or customers interact with the cloud via horizon(Graphical User Interface), OpenStack command line interface(CLI), several cloud management tools(Rightscale, Enstratius etc) and also by making the REST API calls directly to various services.

All OpenStack services authenticate and authorize using a common identity service known as Keystone before performing any of the requested API action by the user.[Opea]

3.6 Existing System

Anomalies in the OpenStack data are monitored using a prometheus exporter which scrapes the data from the nodes and the alerts are configured for these data in the prometheus which is a opensource monitoring tool to configure alerts and these alerts are written manually in a functional query language called PromQL(Prometheus Query Language)[Pro] provided by Prometheus which requires defining the thresholds and setting the values to the metrics. These alerts are monitored using a opensource observability tool known as Grafana and alertmanager in prometheus. For example in order to detect if the nova services are down, a predefined threshold is allotted to the alerts in the alerting rules. So the notification will not be sent to the cloud managers until that threshold is reached. In this case it is considered as a critical alert which is responsible for causing incidents affecting a lot of customers. There is no way to get prior notification to avoid such incidents before the threshold is reached.

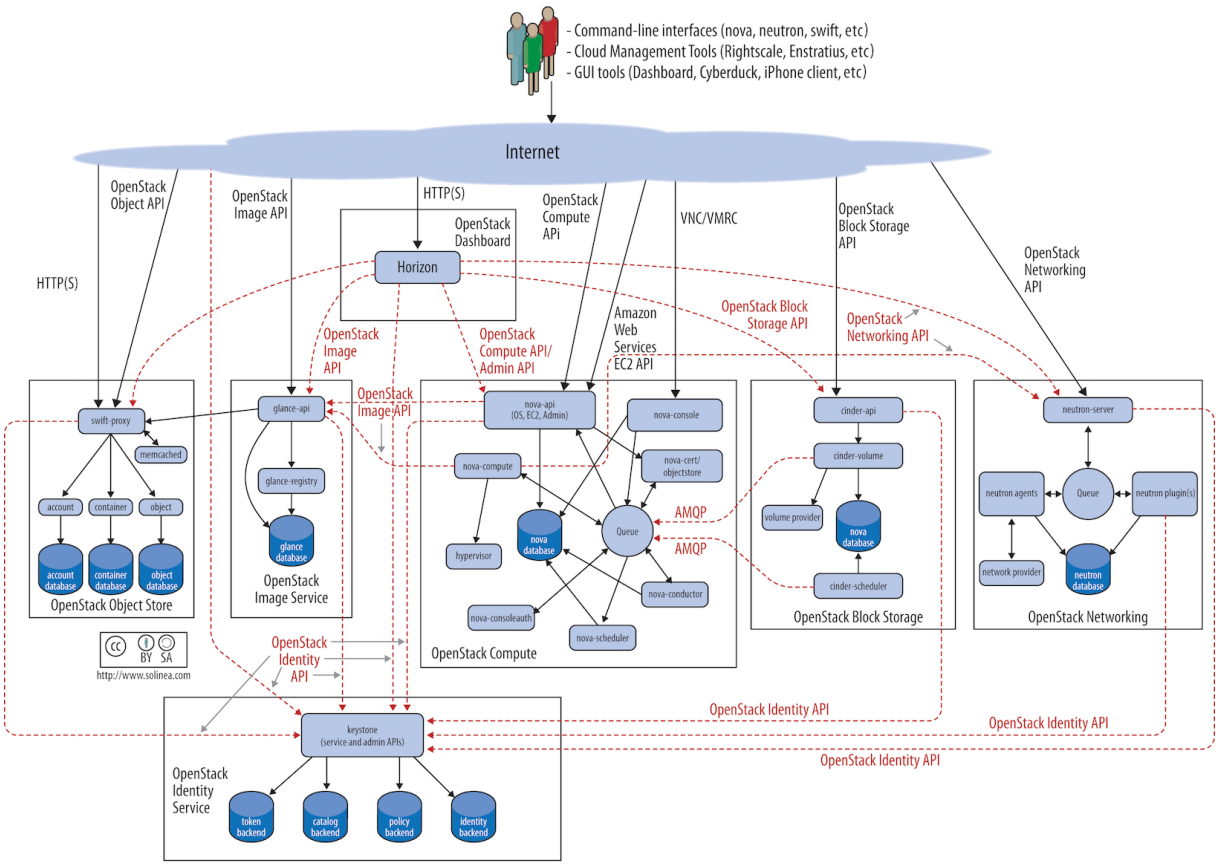


Figure 3.2: OpenStack logical Architecture[Opea]

Once the alert is triggered an SRE is notified via SMS and email along with the alert definition and then they will take the necessary action Or if no such alerts are triggered, the customer will report to the SysEleven cloud support via a support ticket and then the engineer would fix the issue.

Hence a subsequent time would have been passed since the occurrence of the issue and to providing the fix. There is no damage control here. The actions are taken only when the damage occurs and all this is a traditional way of monitoring the system and all the rules are defined by experts which are based on the prior knowledge, visualizations and so on which are effort intensive.

4 Literature Survey

In order to answer the above mentioned research questions, it is very important to study the research papers consisting of existing work done on anomaly detection in the cloud infrastructure. Using the existing approach a best fit solution could be drawn or can be improved for achieving the best results. With this background information re-investigating the existing approaches could be avoided and can concentrate on developing a solution for the defined learning problem based on the DSR.[Hev07]

The literature survey is concentrated on the solutions that use logs and Machine Learning(ML) to detect the anomalies. Hence a systematic research has been carried out and it has been categorized in to two criterias which aims to answer the research questions in depth as the logs produced by different software have different formats. The first criteria focuses on the existing work for detecting the anomalies in the cloud by using the logs in general and also use the ML algorithms to detect the anomalies. The second search criteria includes the research done on the anomaly detection in OpenStack using ML. the research mainly focus on the training model used for the experiment and evaluation criteria.

Key word search like "Anomaly Detection" is used to find the research papers within the following database: IEEE, ACM, Research Gate. Several conference papers, journals, surveys and articles were discovered and they are selected based on the criteria defined as the results yielded also included the approaches without the ML. Several research were conducted and have used several supervised algorithms and hence these are excluded as the dataset is too large and having a labeled dataset for this is not possible. Hence such research papers are excluded from the survey. The Abstract of the resulted papers are scanned and picked the most relevant ones for the study. The table Table 4.1 shows the number of papers selected based on the criterias defined.

- **Selection Criteria**
  - Research that uses Machine Learning
  - Research that uses Unsupervised Learning algorithms
  - Research that uses OpenStack dataset
  - Studies that have used experiments to draw the results and conclusions
- **Exclusion Criteria**
  - Studies that uses other approaches other than the ML
  - Studies that did not focus on the log parsing
  - Studies that use Supervised Learning algorithms

The query used in search of research papers is "Log Based Anomaly Detection in Cloud" AND "Log Based Anomaly Detection in OpenStack" because the query results in the anomaly detection with the focus research done in Cloud and also on the cloud that is dependent on OpenStack. The research is focused on the artefact defined Chapter 2.

Table 4.1: Literature search result for the query "Log Based Anomaly Detection in Cloud" AND "Log Based Anomaly Detection in OpenStack"

Source	No of Hits	After Selection and Exclusion Criteria
IEEE Explorer	81	22
ACM	15	5
Research Gate	30	5

4.1 Existing Anomaly detection in Cloud

Many experiments have been conducted so far to detect the anomalies in the cloud environment. Such experiments involve the usage of supervised algorithms, semi-supervised algorithms and also unsupervised algorithms to detect anomalies.

The drawback of using the supervised algorithms is the need of class labels and class imbalance problem. Cloud environment generates several logs everyday from different components and defining a class label to determine the normal and anomalous data for the huge number of data is a tedious and time consuming task. For this reason this thesis does not concentrate towards the supervised algorithms as the logs generated by the OpenStack components are large in number. A number of logs generated just for a nova-api service on an average is approximately 70,000+ as shown in the figure Figure 4.1

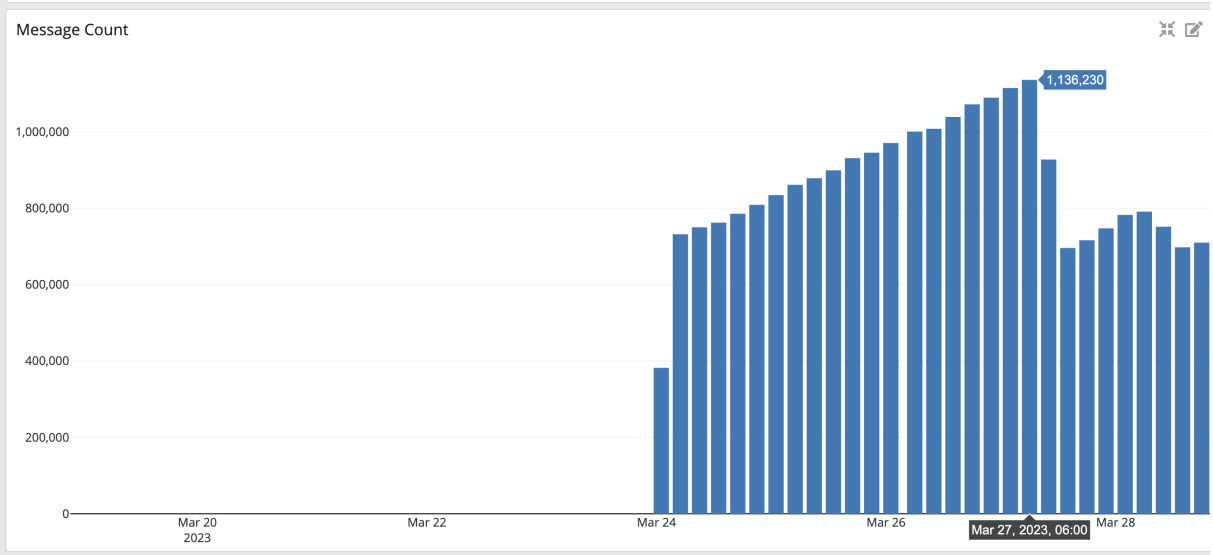


Figure 4.1: Nova logs Generated for a span of 10 days

The semi-supervised algorithm depends on class label as well as it requires a subset of the data to be pre-classified as normal or anomalous by domain experts which again will arise an issue when the large amount of data is being generated and how many such labels needs be defined when lots of components in the cloud are involved.

Because of the above drawbacks this thesis concentrates on unsupervised algorithms for the detection of anomalies in the OpenStack logs in SysEleven Stack. The main reason to choose this is that there is no need of predefined class labels to train the models, there is no fear of class imbalance problem as the data will be segregated into normal or anomalous in the end result.

Based on the above mentioned conditions the following literature surveys are listed in the table Table 4.3.

Table 4.2: Literature search result for the query "Log Based Anomaly Detection in Cloud" AND "Log Based Anomaly Detection in OpenStack"

Title	Authors	Year	Reference
Learning-Based Anomaly Cause Tracing with Synthetic Analysis of Logs from Multiple Cloud Service Components	Yuan, Yue and Anu, Han and Shi, Wenchang and Liang, Bin and Qin, Bo	2019	[YAS+19]
A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks	Garg, Sahil and Kaur, Kuljeet and Kumar, Neeraj and Kaddoum, Georges and Zomaya, Albert Y. and Ranjan, Rajiv	2019	[GKK+19]
Toward an Online Network Intrusion Detection System Based on Ensemble Learning	Hsu, Ying-Feng and He, ZhenYu and Tarutani, Yuya and Matsuoka, Morito	2019	[HHTM19]
Anomaly Detection and Classification using Distributed Tracing and Deep Learning	Nedelkoski, Sasho and Cardoso, Jorge and Kao, Odej	2019	[NCK19a]
Log Message Anomaly Detection and Classification Using Auto-B/LSTM and Auto-GRU	Farzad, Amir and Gulliver, T. Aaron	2019	[FG19]

Continued on next page

Table 4.2 – Continued from previous page

Title	Authors	Year	Reference
LogTransfer: Cross-System Log Anomaly Detection for Software Systems with Transfer Learning	Chen, Rui and Zhang, Shenglin and Li, Dongwen and Zhang, Yuzhe and Guo, Fangrui and Meng, Weibin and Pei, Dan and Zhang, Yuzhi and Chen, Xu and Liu, Yuqing	2020	[CZL+20]
Anomaly Detection from System Tracing Data Using Multimodal Deep Learning	Nedelkoski, Sasho and Cardoso, Jorge and Kao, Odej	2019	[NCK19b]
HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log	Huang, Shaohan and Liu, Yi and Fung, Carol and He, Rong and Zhao, Yining and Yang, Hailong and Luan, Zhongzhi	2020	[HLF+20]
Anomaly detection in cloud network data	Yasarathna, Tharindu and Munasinghe, Lankeshwara	2020	[YM20]
MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures	Arthur Vervaet	2021	[Ver21]
Log-Based Anomaly Detection With Robust Feature Extraction and Online Learning	Han, Shangbin and Wu, Qianhong and Zhang, Han and Qin, Bo and Hu, Jiankun and Shi, Xingang and Liu, Linfeng and Yin, Xia	2021	[HWZ+21]

Continued on next page

Table 4.2 – Continued from previous page

Title	Authors	Year	Reference
Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection	Chen, Zhuangbin and Liu, Jinyang and Gu, Wenwei and Su, Yuxin and Lyu, Michael	2021	[CLG+21]
Anomaly Detection in a Large-Scale Cloud Platform	Islam, Mohammad Saiful and Pourmajidi, William and Zhang, Lei and Steinbacher, John and Erwin, Tony and Miranskyy, Andriy	2021	[IPZ+21]
Fast and Reliable DDoS Detection using Dimensionality Reduction and Machine Learning	Ashi, Zain and Al-Fawa'reh, Mohammad and Aburashed, Laila and Qasaimeh, Malik	2021	[AAFAQ21]
A Systematic Review on Anomaly Detection for Cloud Computing Environments	Hagemann, Tanja and Katsarou, Katerina	2021	[HK21]
Machine Learning Based Anomaly Detection of Log Files Using Ensemble Learning and Self-Attention	Fält, Markus and Forsström, Stefan and Zhang, Tingting	2021	[FFZ21]
ConAnomaly: Content-Based Anomaly Detection for System Logs	Lv, Dan and Luktarhan, Nurbol and Chen, Yiyong	2021	[LLC21]
A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest	Zou, Zhuping and Xie, Yulai and Huang, Kai and Xu, Gongming and Feng, Dan and Long, Darrell	2022	[ZXH+22]
Log-based Anomaly Detection Without Log Parsing	Le, Van-Hoang and Zhang, Hongyu	2022	[LZ22]

Continued on next page

Table 4.2 – Continued from previous page

Title	Authors	Year	Reference
Cloud-based multiclass anomaly detection and categorization using ensemble learning	Shahzad, Faisal and Mannan, Abdul and Javed, Abdul Rehman and Almadhor, Ahmad and Baker, Thar and Al-Jumeily Obe, Dhiya	2022	[SMJ+22]
LogLS: Research on System Log Anomaly Detection Method Based on Dual LSTM	Chen, Yiyong and Luktarhan, Nurbol and Lv, Dan	2022	[CLL22]
Anomaly Detection in Log Files Using Selected Natural Language Processing Methods	Ryciak, Piotr and Wasielewska, Katarzyna and Janicki, Artur	2022	[RWJ22]
DeepSyslog: Deep Anomaly Detection on Syslog Using Sentence Embedding and Metadata	Zhou, Junwei and Qian, Yijia and Zou, Qingtian and Liu, Peng and Xiang, Jianwen	2022	[ZQZ+22]
ADBench: Anomaly Detection Benchmark	Songqiao Han and Xiyang Hu and Hailiang Huang and Minqi Jiang and Yue Zhao	2022	[adb]
Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs	Sánchez-Zas, Carmen and Larriva-Novo, Xavier and Villagrá, Víctor A. and Rodrigo, Mario Sanz and Moreno, José Ignacio	2022	[SZLNV+22]

4.2 Machine Learning In Anomaly Detection In cloud

Based on the literature review, several techniques were used in combination and individually to detect anomalies in the cloud environment which involved techniques of Natural

Language Processing(NLP) and also the Machine Learning.

In the [GKK+19], the detection of anomalies related to the network security by using the Grey wolf optimization(GWO) and the improvised GWO which is used for the multi-objective feature extraction that aims to extract subset of features with lowest error rate and Convolution Neural Network(CNN) for the anomaly detection in hybrid model.

In the [NCK19a] anomaly detection is done based on the distributed tracing records containing the information about availability and response times. The method uses threshold setting and post-processing in order to reduce the false positives. The Gated Recurrent Networks(GRU) along with the autoencoder is used for anomaly detection. In [NCK19b] multi-modal approach is used to train the data only on the normal logs and stacked up layers of LSTM similar to [NCK19a].

In [Ver21] a MoniLog is presented which also uses LSTM along with PCA as it uses a loop to forward the output and its structure is efficient to learn sequential patterns. Log parsing can be done using the available online parsing techniques and a comparative study of some of the online parsing techniques such as Drain, IPLoM, Log-Cluster and so on has also been experimented in the study. The experiment is further extended to cluster the detected anomalies to the priority levels of low, medium and high.

To address the issue of out of vocabulary, semantic misunderstandings and loss of important information that occur due to the log parsing errors, NeuralLog [LZ22] which extract the semantic meaning of raw and unstructured log messages and represents them as vectors. These semantic vectors are then used for training the Transformer-Based model to classify the data to find anomalies. This uses BERT to encode the semantic meaning of log messages. LogLS is proposed with the dual LSTM model which is trained on the log vectors which is similar to the [HLF+20].In [YM20] OCSVM and Autoencoders are tested on Yahoo network dataset in order to detect anomalies and they claim that neural network methods works better than kernel based methods.

Transfer learning approach has been used in [CZL+20] to detect the anomaly and to address the issues of different formats of logs and also reduce the noises in log sequences which are anomalous and is compared against both the supervised and unsupervised algorithms and it outperformed them.

The log template is a semantic-based approach which converts log templates in to vectors using word vectors and then the model is trained on those vectors [HLF+20].

Robust Feature Extraction(RFE) involves using NLP to identify and extract meaningful patterns and relationships within log data, while filtering out irrelevant or noisy information. By doing so, RFE can help to create a cleaner and more structured dataset that can be more easily analyzed and used to train anomaly detection models [HWZ+21]. In [AAFAQ21], the anomaly detection is used for DDOS detection is discussed using the PCA for the dimensionality reduction.

According to the survey in [HK21] the widely used Deep learning methods are MLP, LSTM and Autoencoders where as the classic ML algorithms used are Isolation Forest, LOF, K-Nearest Neighbors, Support Vector Machines, Random Forest, Decision trees. In [GBK+22] a comparative study of 6 algorithms were discussed and the algorithms used are Deep Ant, LSTM, OCSVM, K-Means, DBSCAN, IForest. In [IPZ+21] anomaly

detection for large scale dataset has been developed with the concentration on the GRU and LSTM.

Content based anomaly(ConAnomaly) can be used to get the semantic information from the log message in to the log sequential anomaly detection [LLC21]. Containerization has become a important solution for virtualization in cloud computing and the [ZXH+22] proposed an online container anomaly detection using Isolation Forest by assigning a weights to individual metrics and instead of the random feature selection, weighted feature selection is used in Isolation Forest.

The Natural Language Processing provides different techniques such as word2vec, GloVe and so on which provides dependency of detection on the semantic meanings [RWJ22]. In natural language processing (NLP) tasks, word embedding and sentence embedding have become critical techniques that are widely used. The primary objective of these techniques is to improve the representation of textual data, enabling more accurate and effective analysis and processing of natural language.

DeepSyslog [ZQZ+22] also takes numerical metadata into consideration and which uses sentence embeddings of logs. The survey of anomaly detection algorithms gave more precise conclusion on the state of the art algorithms that could be utilized in the thesis [adb]. The evaluation of traditional algorithms were presented in the [SZLNV+22] which gave the comparison of the performance and suitable preprocessing is done along with one-hot encoding.

The above research papers gave a consolidated information on the experiments and techniques that can be applied to efficiently design a system to detect anomalies.

4.3 Ensemble Methods in Anomaly Detection

In [HHTM19] the stacked ensemble learning is used along with Autoencoders to detect the anomalies caused in the networks by making use of only normal logs to let the model be sensitive towards the attack or anomalies. The usage of stacked ensemble learning combines different algorithm which would analyze the data and capture the information from different perspective. The comparative study on the classic ML models is shown along with the approach of the ensemble model and it resulted in the high accuracy and recall in comparison to the classical ML models. System logs typically contain a wide range of information, including both normal system behavior, performance metrics, as well as unexpected anomalous events such as errors or failures. This means that when analyzing logs, it can be challenging to distinguish between expected and unexpected behavior [FFZ21]. The experiment has been carried to evaluate the behaviour of ensemble learning to detect the anomalies in [FFZ21], where word embedding are generated also uses the self-attension transformer to find the relationship among different words and due to the high variance of the model performance, bagging ensemble technique is used where each base model would vote on the prediction result and the prediction with highest votes is chosen as final prediction. The Cloud-based anomaly detection(CAD) is presented which uses deep learning algorithm to create an ensemble machine learning model to detect the anomalies [SMJ+22] in network.

The ensemble technique can capture the different angles as the combination of multiple algorithms are used in comparison to the individual algorithm that sticks to the single perspective.

4.4 Existing Anomaly detection on OpenStack logs

In the existing experiments, Isolation forest, LSTM(Long Short-Term Memory), GRU(Gated Recurrent Units) along with the AutoEncoders, OSVM(One class support vector machine), KNN(K-nearest neighbors) are used for the detection

Table 4.3: Literature search result for the query "Log Based Anomaly Detection in Cloud" AND "Log Based Anomaly Detection in OpenStack"

Title	Authors	Year	Reference
An Approach to Cloud Execution Failure Diagnosis Based on Exception Logs in OpenStack	Yuan, Yue and Shi, Wenchang and Liang, Bin and Qin, Bo	2019	[YSLQ19]
LogSayer: Log Pattern-driven Cloud Component Anomaly Diagnosis with Machine Learning	Zhou, Pengpeng and Wang, Yang and Li, Zhenyu and Wang, Xin and Tyson, Gareth and Xie, Gaogang	2020	[ZWL+20]
Log Message Anomaly Detection with Oversampling	Farzad, Amir and Gulliver, T. Aaron	2020	[FG20a]
Experimentations with OpenStack System Logs and Support Vector Machine for an Anomaly Detection Model in a Private Cloud Infrastructure	Akanle, Matthew and Ajala, Sunday and Adetiba, Emmanuel and Moninuola, Funmilayo and Akande, Victor and Badejo, Joke and Adebiyi, Ezekiel	2020	[AAA+20]
Unsupervised log message anomaly detection	Farzad, Amir and Gulliver, T. Aaron	2020	[FG20b]

Continued on next page

Table 4.3 – Continued from previous page

Title	Authors	Year	Reference
Improving Performance of Log Anomaly Detection With Semantic and Time Features Based on BiLSTM-Attention	Li, Xinqiang and Niu, Weina and Zhang, Xiaosong and Zhang, Runzi and Yu, Zhenqi and Li, Zimu	2021	[LNZ+21]
Global Monitor using SpatioTemporally Correlated Local Monitors	Yadav, Geeta and Paul, Kolin	2021	[YP21]

The Machine Learning techniques are used for anomaly detection in OpenStack Logs. [YSLQ19] exploits log level ERROR for the diagnosis of the failure of the cloud operating system in order to find the root cause for the exceptions. Here the logs extracted are stored by creating a profile database and the exception logs are compared with the historical exception logs and these logs are modeled using word2vec. LogSayer is a log pattern driven diagnosis for anomalies using machine learning which presents the offline and online training for anomaly detection [ZWL+20]. In [FG20a] random oversampling and the Synthetic Minority Oversampling Technique (SMOTE) is used to increase the minority samples along with the deep learning algorithms to detect the anomalies. The SVM and Tf-Idf is used on the publicly available OpenStack corpus to evaluate the performance of SVM for anomaly detection [AAA+20]. The sentence embeddings are extracted for the training using BERT model in [LNZ+21] which is then trained with the classic LSTM and Bi-LSTM. In [FG20b] unsupervised anomaly detection method is used in combination with two autoencoders. This is used as a base for comparison in this thesis. The logs are preprocessed to get the unique words as features using one-hot encoding and data is split and a part of data is trained with first autoencoder and rest of the data is used to extract the features and Isolation forest is trained on it. The resulting positive and negative data is used to the second autoencoder. The threshold is set based on the negative logs in the test set and the mean is compared against it to predict the anomaly. The algorithms outperformed several traditional algorithms like K-means, GMM, BGM, EEnvelope, Local Outlier Factor, One Class-Support Vector Machine.

From the above literature survey it is clear that Machine Learning is a suitable, systematic, cost effective and efficient solution in terms of performance for the anomaly detection in comparison to the traditional way of using the monitoring tools that has case by case alerting rules defined to detect the anomalies. The below table 4.4 shows a list of techniques, algorithms used in the literature survey along with the evaluation methods. The widely used algorithms are LSTM, Isolation Forest, Autoencoders, K-Means, LOF, SVM for supervised learning. The state of the art COPOD and ECOD algorithms have not been tested to the best of our knowledge and hence the thesis aims to test the algorithms Isolation Forest, LSTM, LOF, COPOD and ECOD.

Table 4.4: List of Parsing Techniques, Algorithms, Evaluation used in the literature survey

Parsing	**Algorithms**	**Evaluation**	**OpenStack**
Log Parser	Isolation Forest, LSTM, K-Means, LOF, GRU, Bi-LSTM, Autoencoder, K-NN, OC-SVM, Decision Trees, Random foreest, SVM, Transformers, CNN, Ensemble	Quantitative, Qalitative and Quantitative for Isolation forest, LSTM	Yes
Custom Parser	LSTM, Isolation Forest, Autoencoders, Transformers	Qualitative	Yes
No Parser	LSTM, Transformer	Qualitative	NO

5 Theoretical Background

5.1 Log Parsing

Log Parsing is the first step in case of the log analysis. This includes parsing of raw, unstructured logs from different sources in the cloud to a structured format in order to analyze and train the models for anomaly detection.

Several log parsing algorithms are available for this use. Such log parsing algorithms include Drain algorithm, IPLoM algorithm which are widely used for the parsing and are claimed to parse the log files within a short time. This parsing techniques have been tested on the OpenStack logs as well.

5.1.1 Drain

Drain is a fixed depth tree based algorithm used for log parsing which uses a regular expressions based on the domain knowledge to preprocess them. Their main motive is to convert the raw, unstructured logs in to structured logs.[HZZL17]

- Step I: Step I is to preprocess the raw log messages by using simple regular expression based on the knowledge on the domain. In this step the tokens are matched instead of the whole log message using the regular expression.
- Step II: This step is to perform a search by log message length with an assumption that "messages with same log event might have same log message length" [HZZL17]. The log messages that has the same log event but different log message length will be handled using additional processing after the application of Drain method.
- Step III: The third step is to search the logs by making use of the preceding tokens. In this case the Drain traverse from 1st layer node from the step II to a leaf node based on an assumption that "the token in the beginning position are likely to be constants"[HZZL17] and the token that does not have digits are selected here.
- Step IV: The fourth step is to search using the token similarity. These leaf nodes consists of log groups. In order to pick the log group from the list in the leaf node, a similarity is calculated between the message and log event using the formula Equation 5.1.1

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

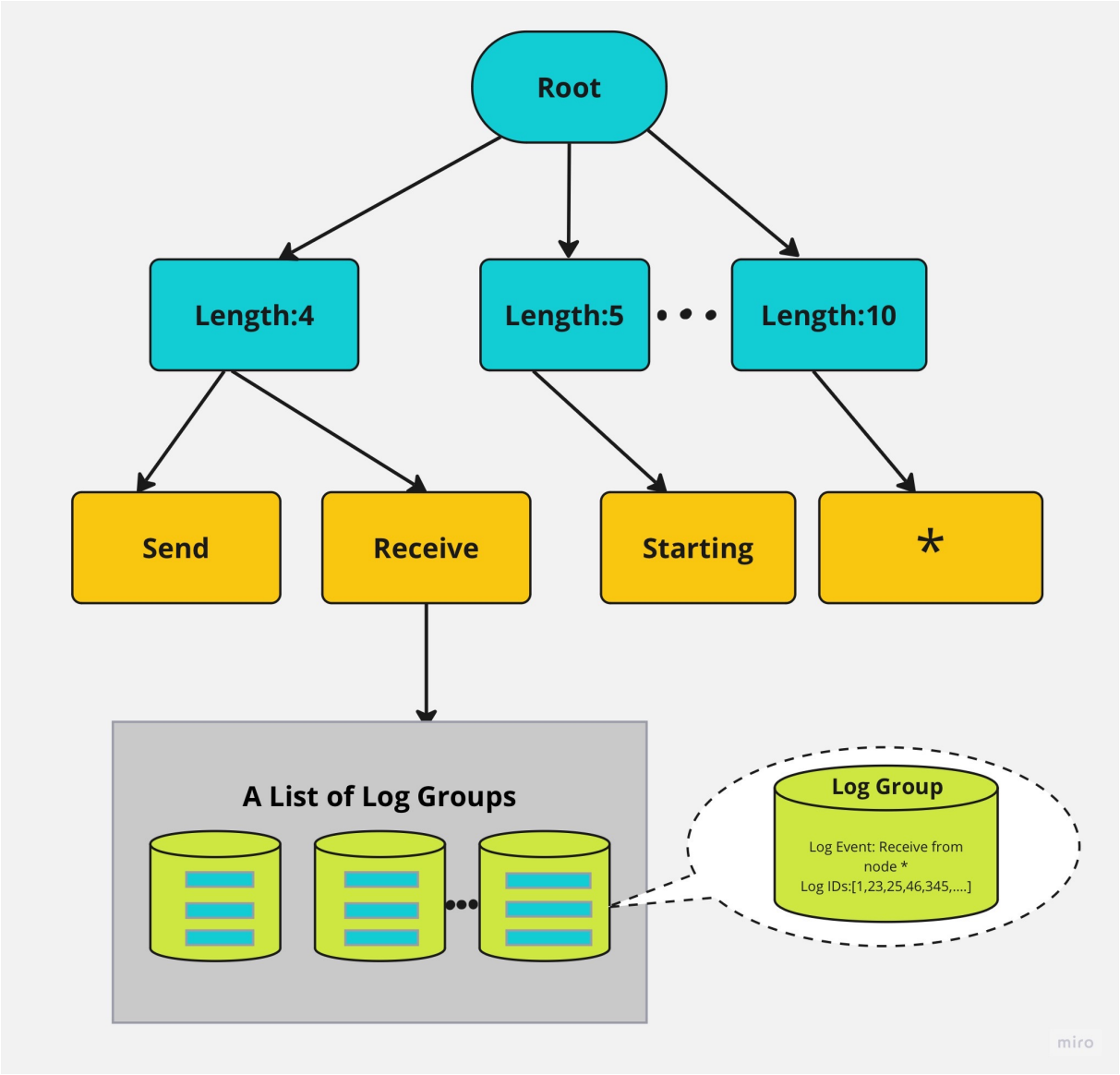


Figure 5.1: Structure of the parse tree in Drain with depth=3[HZZL17]

where, seq1 and seq2 are log messages and log event; seq(i) is the i-th token of the sequence and n is the log message length of the sequences [HZZL17].

The function equ in the is calculated using Equation 5.1.1

$$equ(t_1,\ t_2) = \begin{cases} 1 & \text{if } t_1 = t_2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where, t1 and t2 are two token. seq1 and seq2 are log messages and log event; seq(i) is the i-th token of the sequence; n is the log message length of the sequences. Once the log group is discovered, it is compared with the predefined threshold st and if simSeq is greater than or equal to st then the Drain returns the group as the most suitable log group[HZZL17].

- Step V: The final step is to update the log parser. Log ID of the current log message

is added to the most suitable log group that has been formed in the step 4 or the new log group. Drain will scan for the token in same position and update the token position by wildcard in the log template if the two tokens are different. [HZZL17]

Drain works based on the assumptions stated in the Step II and Step III which is not the case always. There are also logs observed which are just starting with plain sentences and Drain will discard those logs with out processing them. Such logs are required to teach the model to better understand the anomaly nature when classifying.

5.1.2 Iterative Partitioning Log Mining

Iterative Partitioning Log Mining(IPLoM) is the state of the art algorithm for the log parsing. The unique characteristics present in the log messages are used for the iterative log parsing[MZHM11].

- Step I: Partition by event size in which the logs are separated into different clusters based on the length with the same assumption as Drain that the logs belongs to one template are of same length.

- Step II: Partition by token position. In this every cluster of logs with the equal length can be represented as m-by-n matrix. This partition is based on the assumption that "the column that has the least number of unique words is the one that contains constants"[MZHM11]. Because of which the split word position will be used in order to partition each of the clusters in order for them to have same word in that position.

- Step III: Partition by search for mapping process selects two columns for further processing by partitioning them based on the mapping relation that exists between them. In order to decide which two columns are selected, the word count of unique words are taken from each column and the columns with most frequent word count are selected. Total 4 mapping relations are considered namely 1-1, 1-M, M-1, M-M. In case of 1-1 relation two columns are used to partition in to the same cluster. In case of 1-M and M-1 the column that contain constants is used to partition logs. The M-M relation will be partitioned in to single cluster[MZHM11].

- Step IV: Log template extraction. In this step IPLoM will process through all the clusters that are formed in the previous step and then it will generate one log template per cluster. For each column, unique word count is taken. The unique word is considered as constant and rest of the words are represented as attributes and are replaced with wildcard in the output[MZHM11].

Similar to Drain the IPLoM also works on the assumption that the logs belonging to one template have same log length. Also the IPLoM discards the plain sentences while parsing the logs thus reducing the uniqueness of data for training.

5.1.3 Regular Expression

Regular expression is a pattern that the input data matches to give out the desired output. Based on the domain knowledge and to extract the useful information out of the log messages regular expressions can be used in a very effective way. The goal is to preprocess all the incoming logs without discarding them and sending it to the training and prediction. But the Drain and IPLoM fail to do this.

In order to address the issue of finding a plausible log parser the decision has been made to write a regular expression that suits the OpenStack component logs such that the unnecessary information can be removed from the logs. Hence to fit the OpenStack logs a parser file can be written and used for initial preprocessing of the data which can be fast enough to preprocess large volumes of data.

5.2 Feature Engineering

Feature Engineering is a very important step in achieving the detection of anomalies in the logs. There are several techniques introduced for finding the suitable features and to extract them for predictions.

5.2.1 Feature Extraction

CountVectorizer

CountVectorizer is a method of extraction of features in a textual data by converting the text in to a numerical data. Since the machines needs numerical data in order to process, all the textual data must be converted in to numerical data. After applying the CountVectorizer, the text is transformed to a sparse matrix. This is passed as an input to the model for training.

Term Frequency-Inverse Document Frequency(Tf-idf)

Tf-idf is a statistical method which depicts the relevancy of a text in relation to the terms obtained in the logs. This is achieved in two different parts in order to provide a relevancy score.

The first part is to find the term frequency. In general, this is the number of times the word occurs in a single sentence.

$$tf(t, d) = number\ of\ occurrence\ of\ t\ in\ d \tag{3}$$

where, d is the document and t is the term/word.

The second phase is the document frequency which is a number of documents d that consists of the word t.

Next is to find the inverse document frequency(IDF) which tests how relevant the word is. The main concentration here is to search the record which is most relevant when there exists a common term in the query. The IDF of a term is found by dividing the number of sentences by number of sentences that consists of a given term t.

$$IDF(t) = log_{10}(\frac{d}{d'}) \tag{4}$$

where, d is the number of sentences and d' is the number of sentences that consists of the word t [DSM19]

The final weight of the word using Tf-idf is given by combining all the parameters. The formulas for finding the Tf-idf is,

$$tf - idf(t,d) = tf(t,d) * log(\frac{d}{d'}) \tag{5}$$

where, the tf-idf of word t is the product of the term frequency tf(t,d) and IDF(t)[DSM19]

The higher the score of the tf-idf of a word t, the higher the relevance of the word for a sentence. [DSM19]

5.2.2 Word Embedding

As said above the machine cannot understand the textual data in order to process them and give the output. Hence these data must be converted in to the numerical format which the model understands. The word embedding are the most used representation of data as an input to the deep learning algorithms. The resulting vector captures the semantic relationship of the word, definitions and so on. Using these semantic relationships the similar or dissimilar words can also be found. The simplest form of embeddings is one-hot encoding. There are others such as Word2Vec and Glove among which Word2Vec and One-hot encoding is used in the thesis for comparison.

One Hot Encoding

One Hot Encoding is used to convert the categorical values in to binary vectors to avoid the model from interpreting that an order exists in the dataset. Even though this is the simplest form of representation, there are few drawbacks such as they fail to capture the characteristics such as similarity of the words and they would make the feature set grow quite large in case of the huge amount of data such as logs.

Word2Vec

Using Word2Vec to represent the word embeddings is more efficient in case of large datasets. It estimates the word's meaning based on its occurrence in the text. It effectively groups the vectors of similar words together. It generates the high-dimensional vector of the words by using the two layer neural networks architecture. These layers are skip-grams and continuous bag of words(COBOW).

- **Continuous Bag Of Words(COBOW)**: It is feed forward neural network model which will predict the target word based on the preceding word and also based on the context of the sentence.
- **Skip-grams**: It is a simple neural network which predicts the probability of surrounding words being present when an input word is given. In general it is the opposite of the COBOW model where it takes the word as an input and predicts the before and after of the input word.

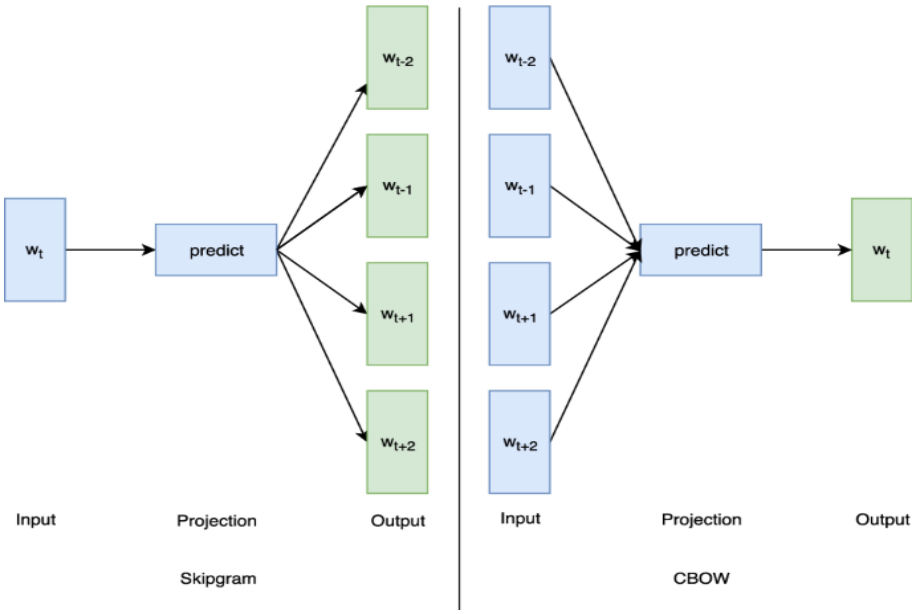


Figure 5.2: Word2Vec model[Fra]

The target word is selected using the rolling window as a parameter. The number of adjacent words that needs to be selected are determined by the size of the window specified. Finally the data that must be trained consists of the pairwise combination of the target words and the other terms that are present in the window specified. The result of the training gives the probability values which shows that higher the value the more likely the term will be present near the target word.

5.2.3 Feature Selection

Based on Domain Knowledge

Domain knowledge is defined as a background knowledge on a topic. Using this knowledge an expert can determine what features contribute to the efficient decision mak-

ing.

Trying to pick features that are suitable for model training only based on the domain knowledge is a tedious task due to enormous amount of feature set based on the unique words present in the text data. Examining each word to be suitable for the decision is very time consuming. But one cannot ignore the fact that domain knowledge is very much needed to determine the contribution of each features selected from the bunch of features from any feature extraction techniques.

In this case knowledge about the OpenStack components such as Nova, Cinder, Neutron, Keystone and so on is very much needed to determine what features contribute towards the detection of anomaly.

5.2.4 Dimensionality Reduction

If the feature space is too big there is always a possibility of potential difficulties such as efficient way to define the relation between the attributes, over-fitting the model and also violating the assumptions that have been made. In order to consider all the features that have been collected and to focus only on the few, the dimension of the feature space must be reduced. This process is called "dimensionality reduction".

Principal Component Analysis

Principal Component Analysis(PCA) comes in to picture when we are dealing with lots of variables in the dataset. PCA is a type of feature extraction technique which combines the attributes in order to reduce the dimension such that the lesser contributing variables could be removed, while still keeping the important information of all the variables. The variables obtained after the PCA are not dependent on each other.

In order to perform PCA, the following steps must be followed,

- Take the dataset D as an input data and represent it into a structure such that every row in the dataset corresponds to the data items and the columns corresponds to the attributes. The dimension of the dataset is determined by the number of columns.
- Next step is to standardizing the data. The attributes with higher variance are considered more important than the attributes with the lower variance. If the importance of the attribute does not depend on the variance of the attribute, then each data item is divided by the standard deviation of the column. The resulting matrix is named as Z.
- Next covariance of a matrix(CV) of Z must be calculated. In order to do so,

$$CV(X,Y) = \sum_{i=1ton} \frac{(X_i - X')(Y_i - Y')}{(n-1)}, \tag{7}$$

  where, X and Y are variables, X' and Y' are the arithmetic mean of X and Y respectively and n is the number of observations[MST+17]

- After obtaining CV, the eigenvectors and eigenvalues should be calculated for the CV of Z. Eigenvector is the direction of the axes with more information and the eigenvalues are the coefficient of these eigenvectors.
- Sort the eigenvalues in the decreasing order and accordingly the eigenvectors should be sorted in the matrix P with eigenvalues. The output matrix is P*
- Calculate the new attributes. The resultant matrix Z* is obtained by multiplying P* and Z and in Z* each observation is considered as the linear combination of original features. The principal components obtained are orthogonal to each other[MST+17].

$$Z* = Z \times P*, \tag{8}$$

- Final step is to drop the unimportant features by keeping only the relevant and important features.

AutoEncoder

AutoEncoder is a unsupervised neural network which is used for representational learning by imposing a bottleneck in the neural network that would result in a compressed form of the original input. This is achieved by the correlation between input features.

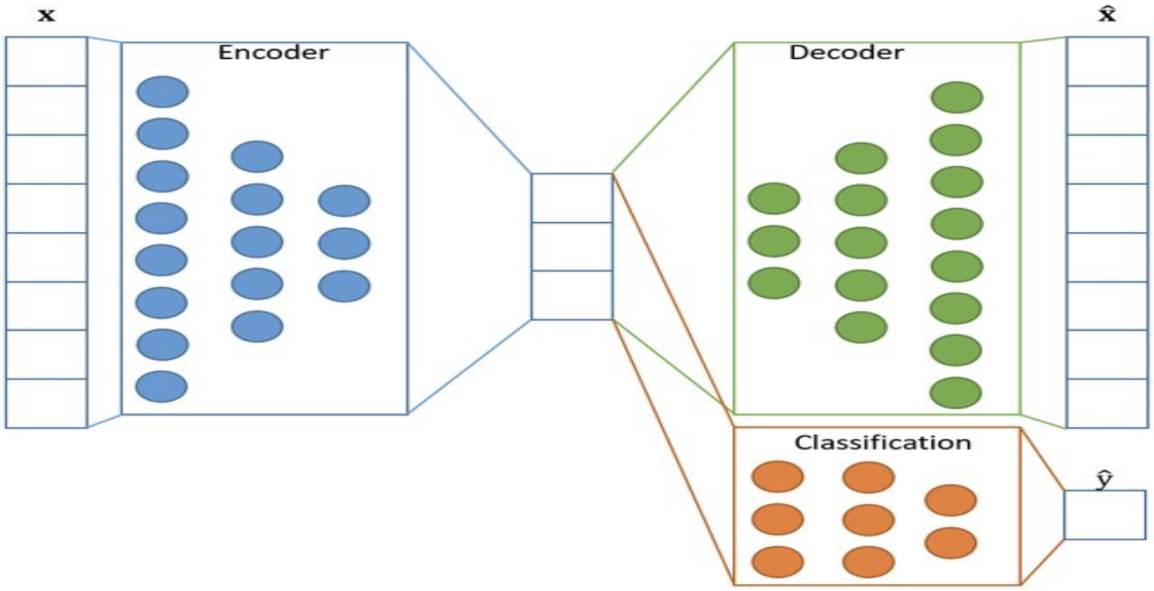


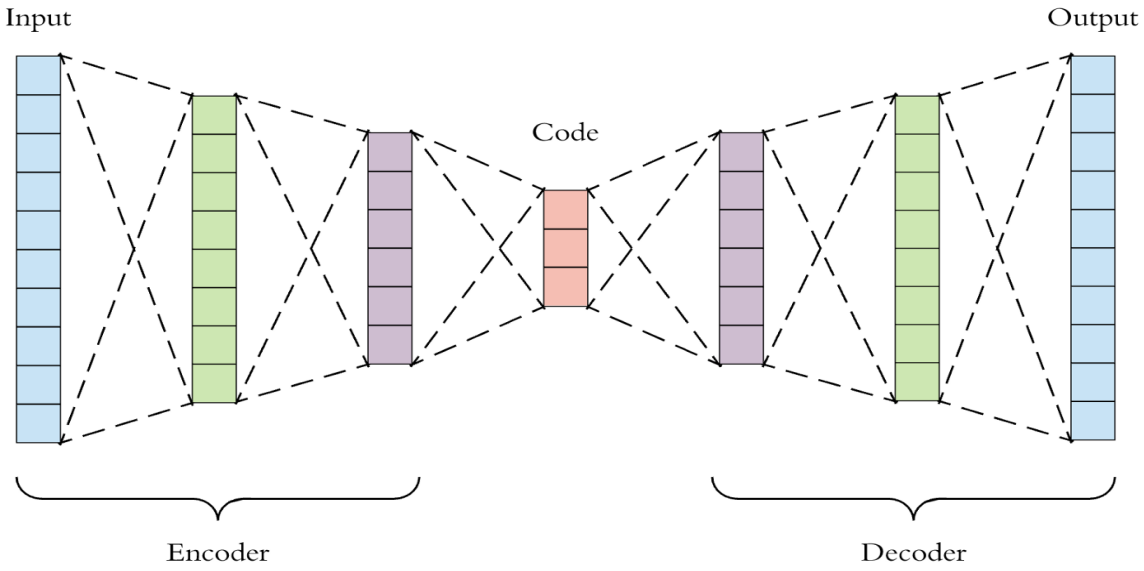
Figure 5.3: AutoEncoder [BKG20]

The reconstruction of X(original input) is done by taking the unlabelled dataset and building it as a supervised learning problem. The resulting output is X‘ which is a reconstruction of X. $(E(X, X‘))$ is the reconstruction error which measures the differences between our original input and the reconstruction. The network is trained and this reconstruction error should be minimum. Without the Code layer, the network would learn to memorize the input values given to it by simply passing it through the network.

The input first passes through the encoder layer(fully connected neural network) and produces the code. Decoder should also have the similar neural network structure and it produces output using the code. The hyperparameters must be set before training and such parameters are code size, number of layers, number of nodes per layer and also the loss function.

AutoEncoders for dimensionality reduction

The number of neurons in the encoder layer will be decreasing and the decoder will be increasing when moving on with further layers. The number of neurons in the bottleneck must be less than the number of features.



While using AutoEncoder for dimensionality reduction the bottleneck layer is extracted and used to reduce the dimensions. This is known as Feature Extraction process.

After training the AutoEncoder, the encoder layer is used to get the dimension in the data. The number of features reduced will be equal to the number of hidden units in the bottleneck

5.3 Algorithms for Anomaly detection

Many ML algorithms are available for anomaly detection which efficiently detects the anomalies in the cloud. These anomalies automates the process. They are classified as supervised, unsupervised, semi-supervised and reinforcement based on the nature of how the training has been done. The classification is briefly explained below.

- **Supervised Learning:** The algorithms that fall under this category requires external supervision to be done. Hence it is called as Supervised Learning. These algorithms uses labelled data for the training. These type of algorithms need the

certain level of expertise in the learning problem to handle them. If they are extremely difficult to use for the large amount of data because they use labelled data for training and labelling huge amount of data such as logs is not a feasible option. Hence they are not being tested in this thesis.

- **Semi-supervised Learning:** This set of algorithms use both labelled and unlabelled data for the training. It is trained in inductive(it is constructed and trained based on an existing labelled training set and predicted on an unseen data using the trained model) and transductive(both the training and testing data has been observed previously and it uses patterns, information that it has encountered during the training to predict the unlabelled data. If a new data point is encountered then the model has to be retrained) fashion. This type of model is less reliable in comparison with the supervised learning and the outcome of the iterations are unstable.

- **Unsupervised Learning:** This algorithm does not require any supervision. It is useful when the human experts does not know what to look for in the data and also it is useful where there are large amounts of data available. It uses techniques on the input data to learn patterns, rules, similarities and cluster the data points. The main advantage of these algorithms is that it does not require any experts to label the data and hence this category will fit perfectly for the large datasets like logs. Hence, algorithms that belong to this category are chosen for anomaly detection.

- **Reinforcement Learning:** These type of algorithms uses feedback mechanism which has been gathered from its own errors. In the process of training it makes series of decisions and it awards its positive decision and assigns penalties to its negative decisions. This way the model tries to take actions that are inclined more towards positive decisions. Even though lot of data can be provided for this type learning in logs, it is computationally costly and it is always difficult to justify why the agent is behaving in a certain way.

The algorithms that have been used in log-based anomaly detection which are common, state of the art and novel in nature are chosen for comparison and such algorithms are Local Outlier Factor, Isolation Forest, LSTM, COPOD, ECOD and ensemble learning. Each of these are explained in detail below.

5.3.1 Local Outlier Factor

Outlier is considered as a data point which does not belong to the rest of the data. It is considered to be a abnormal observation that are there to show erroneous conditions and the data point is said to be "local outlier" when the sample is said to be outlier based on its local neighborhood.

Local outlier factor(LOF) algorithm is used to identify the outliers in the dataset and it will identify such data points using the neighborhood's density. LOF depends on the concept of K-Neighbors and Local reachability density(LRD).

K-Neighbors denoted by $N_k(p)$ is defined as the set of points that lie in or on the circle of radius K-distance(distance between the point and its K^{th})[BKNS00].

Reachability distance(RD) is defined as the maximum of K-distance of object o and the distance between the p and o. The distance can be measured using Euclidean distance, Manhattan and so on[BKNS00]. The formula for reachability distance is given below.

$$reach - dist_k(p, o) = max(K - distance(o), d(p, o)) \tag{9}$$

Local reachability density(LRD) is the average of the reachability distance of p based on the min-points nearest neighbors of p($N_k(p)$). More the average of RD, less the density of the data points that are present i.e, how far the point is from the cluster(nearest). Lesser the value of LRD, the distance between the closest cluster and the point is more.

$$LRD_k(p) = \frac{1}{\sum_{o \epsilon N_k(p)} \frac{reach - dist_k(p,o)}{|N_k(p)|}} \tag{10}$$

where, the local reachability density of an object p is the in-verse of the average reachability distance based on the MinPts-nearest neighbors of p. LOF is the ratio of the average LRD of the neighbors to the LRD of p[BKNS00].

$$LOF_k(p) = \frac{\sum_{o \epsilon N_k(p)} LRD_k(o)}{(|N_k(p)|).(LRD_k(p))} \tag{11}$$

where, nearest neighbors for object p is given by $N_k(p)$

It can be depicted from the above equation if the LOF is less than 1 it is considered as inliner and if it more than 1 it is an outlier. and if LOF is approximately equal to 1 then the density of a point and its neighbors are almost equal.

Long Short-Term Memory

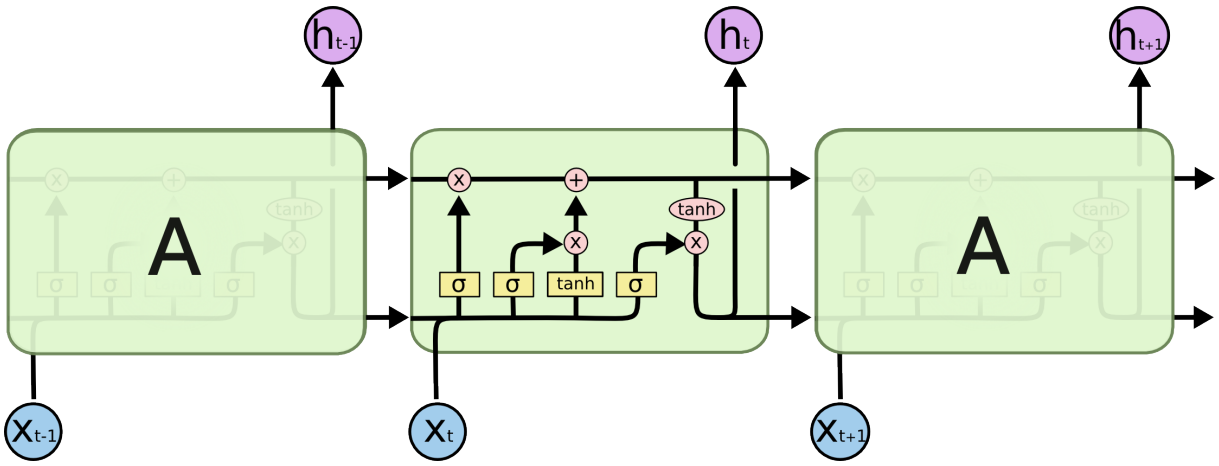
Long short-term memory addresses the issue of long-term dependency problem. They are a type of Recurrent Neural Network(RNN) which allows information to be remembered using hidden state for longer period of time. The LSTM has a four neural netwoks each of which performns a distinct specific tasks and they do not have recurring module as a typical RNN.

First the LSTM must choose what needs to be discarded and it is decided by sigmoid layer called "forget gate layer". If it is 1 then "Completely keep this", if it is 0 "Completely get rid of this". the output of this step is given by the formula,

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{12}$$

where, h_{t-1} is the previous state and x_t is the current input.

Next, decide on the new information the needs to be stored in a cell state. In order to update the value sigmoid layer is used which is called as "input gate layer" and the new candidate values will be created by tanh layer, C_t which is added to the state.



$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{13}$$

$$C_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{14}$$

To update the cell state C_{t-1} to C_t which is a new cell state, multiply f_t to the old cell state C_{t-1} and add $i_t * C_t$.

$$C_t = f_t * C_{t-1} + i_t * C_t \tag{15}$$

Finally what is declared as output is based on the cell state as a filtered version. The sigmoid layer will decide the parts of cell state that should be outputted and then it is passed through tanh layer in order to push the values between -1 and 1 and multiply it with sigmoid gate to output only the parts that have been decided.

$$C_t = o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$o_t * tanh(C_t) \tag{16}$$

5.3.2 Bi-Directional LSTM(Bi-LSTM)

In case of Bi-LSTM, the network remembers both the previous and the next word in the text. In general, they are putting two independent LSTMs in order to preserve information from both posterior and preceding. Bi-LSTMs are proven to understand the context better when compared to the LSTM as it propagates only backward. Bi-LSTMs are useful where the context of the input is also needed and the information flows from backward to forward and vice versa using two hidden states which makes it understand the context of the text better. The Figure 5.4 shows the architecture of Bi-LSTM.

5.3.3 Isolation Forest

Isolation Forest is a type of an ensemble approach which has been mainly designed to detect anomalies. It is based on the Decision trees and is used for outlier detection.

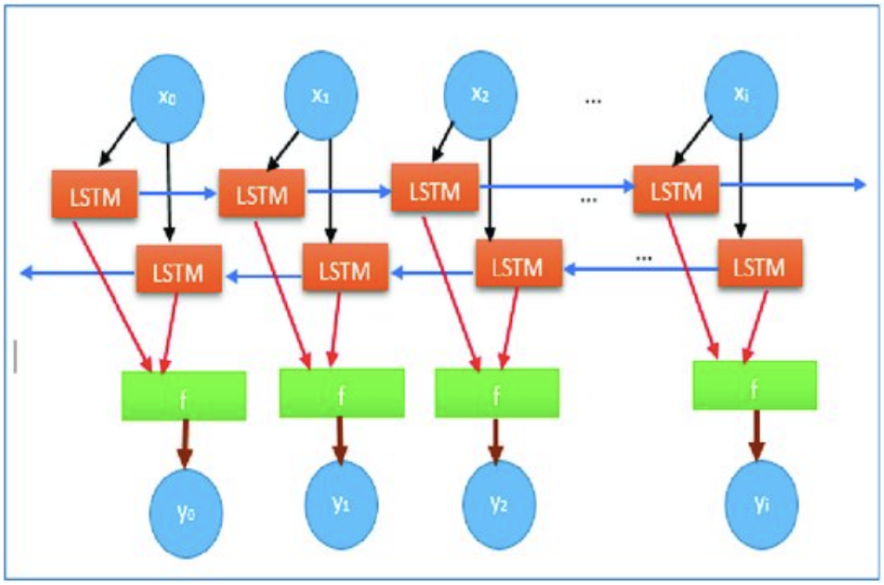


Figure 5.4: Bi-LSTM Architecture[HO21]

Since no labels will be specified here, it is considered to be an unsupervised algorithm. The samples that travel deep in to the tree are less likely to be outliers as they require more cuts to isolate them. Isolation forest consists of number of isolation trees which are divided into various attributes.

Isolation forest selects a random dimension and splits the data along that dimension randomly. "Isolation" means separating a data point from rest of the data points since they are very few points different from the normal occurrence.[LTZ08]

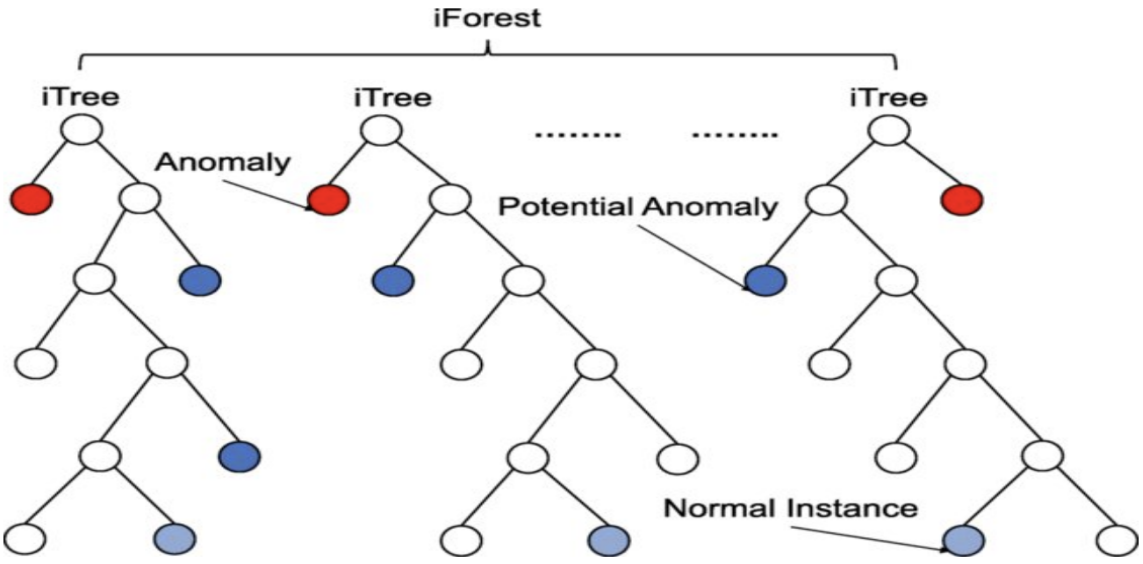


Figure 5.5: Isolation Forest[RFA21]

The itrees will be constructed by parsing the training data until the sample is isolated. Given an input which involves a dataset X, number of trees, sub-sampling size, the height limit of the tree is set by sub-sampling size and growing the trees til the average tree height

is because the importance is only given to the samples that have a path length shorter compared to the average path length[LTZ08] and these points are the ones considered as anomalies.

The model training is complete after the creation of the iTrees. In order to score, a data point needs to be traversed through all the trees that are trained earlier. An anomaly is indicated by -1 and 1 is assigned to normal points based on the depth of the tree to the data point and the contamination parameter set which is a percentage of the anomalies present.

There are few limitations to Isolation Forest:

- The assignment of anomaly score is based on contamination defined. In order to define the contamination one should have a knowledge about the dataset and the percentage of anomalies that might be present.
- Because of the way the branching is done, the model is suffered from bias.

5.3.4 Copula Based Outlier Detection

Copula Based Outlier Detection(COPOD) is one of the recent algorithms which has several key features like no hyperparameter tuning(this is a difficult task as the true labels are rare or unknown in the outlier detection), fast and computationally efficient, easy to visualize anomalies, scales to high dimensional datasets, which makes the algorithm to standout. The marginal distribution can be separated form the dependency structure of the multivariate distribution using Copula function[LZB+20].The COPOD has been trained on benchmark datasets and in most cases it has outperformed and also one of the fastest outlier detection algorithm.

COPOD algorith can be formally defined in three stages. For each column, the left tail empirical cumulative distribution function(ECDF) is fitted and also fit the right ECDF and then compute skewness coefficient.

The ECDF $F`(x)$ can be computed as,

$$F\hat{}(x) = P((, x]) = \frac{1}{n} \sum_{i=1ton} I(X_i x) \tag{17}$$

The skewness vector b= $[b_1, ..., b_d]$, where [LZB+20]

$$b_i = \frac{\frac{1}{n} \sum_{i=1...n} (x_i - x_i`)^3}{\sqrt{\frac{1}{n-1} \sum_{i=1...n} (x_i - x_i`)^2}^3} \tag{18}$$

Using the above function compute the left tail and the right tail empirical copula and then compute the skewness corrected empirical copula values for every sample in the dataset[LZB+20].

Last step involves computing the anomaly score for each row based on the values of left tail, right tail and skewness-corrected probabilities. The smaller copula values result in a larger negative-log values. Anamoly scores lie between $(0, \infty)$ and can be compared

directly. In order to get the outlier predictions using the COPOD, two ways are possible,

- Threshold can be set on the outlier score and any data point that exceeds the threshold is considered as anomaly.
- Select to the top-α percentile of the outlier scores.

Since the log data has a huge volume of data it can be useful to see if COPOD out performs compared to other algorithms.

So far the COPOD is never tested on the Cloud logs especially for the OpenStack and hence would make a good candidate to test its performance.

5.3.5 Empirical-Cumulative-distribution-based Outlier Detection

This algorithm is an unsupervised outlier detection using ECDF which make use of the information about the distribution of the data to determine if the data is outlier or not. Similar to the COPOD, ECOD also estimates the ECDF for each feature in the data separately. The model would fit the joint ECDF over all the attributes and this is computationally expensive and as the number of features grows the rate of convergence for estimating the joint Cumulative Distributed Function slows down. Hence ECOD makes the assumption that all the features are statistically independent.

First step is to compute the left and right tails ECDF for each row of data.

$$F^{`(j)}_{left}(z) = \frac{1}{n} \sum_{i=1ton} (X_i)^{(j)} <= z \tag{19}$$

$$F^{`}{}_{right}(x) = \frac{1}{n} \sum_{i=1ton} (X_i)^{(j)} >= z \tag{20}$$

Then for the jth feature distribution, skewness coefficient must be computed. This is done using the formula,

$$\gamma = \frac{\frac{1}{n} \sum_{i=1ton} (X_i)^{(j)} - X^{`(j)})^3}{[\frac{1}{n-1} \sum_{i=1ton} (X_i^{(j)}) - X^{`(j)2})^{3/2}]} \tag{21}$$

The fit function is iterated via the for loop, hence it can be parallelized for faster fitting. In order to find the outlier three values namely O_left which is the sum of the negative log of the left tail and O_right is the sum of the negative log of the right tail and O_auto is the sum of the left or the right probability based on the skewness coefficient.The final outlier scores is computed buy taking the maximum of all the 3 outlier values.

$$OutlierScores = max(O_{left}, O_{right}, O_{auto}) \tag{22}$$

The outlier score lies between (0, ∞).

- Set the threshold on the outlier score and the row that exceeds the threshold is considered as anomaly.
- Select the top-kth percentile of the outlier scores
- Hand pick the threshold

ECOD does not require hyperparameter tuning, this is an important feature as the hyperparameter tuning is difficult as the true labels are unknown in the unsupervised learning. The algorithm is very fast and computationally efficient and the time complexity scales linearly.

5.4 Evaluation Metrics

The models can be evaluated based on both the quantitative and qualitative assessment along with the amount of compute resource required. The dataset is split in to training and testing or can have an unseen evaluating set. The model is trained on training set and this trained model can be used to predict the test set. The predicted labels can be evaluated based on the expert opinion and also on the quantitative measures such as Accuracy, F1 score, Precision, Recall. Using multiple metrics to evaluate the model is really important as they might perform well on one metric and might perform poorly on the other metric. Each of these metrics are explained below.

- **Accuracy:** It is the proportion of correctly classified labels among the cases used in examination by the model. This can be used when there is no threat of class imbalance problem[Man22].

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{23}$$

- **Precision:** It is the value that gives what percentage of the predicted positive values are actually positive. Generally it is an accuracy of how many predicted positive are truly positive[Man22].

$$Precision = \frac{TP}{(TP + FP)} \tag{24}$$

- **Recall:** Recall value tells what percentage of true positives are correctly classified[Man22].

$$Recall = \frac{TP}{(TP + FN)} \tag{25}$$

- **F1-Score:** F1-score is the harmonic mean of precision and the recall. If the precision is low, then the F1-score will also be low and same is for recall too[Man22].

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{26}$$

- **Support:** The support denotes the number of each class label in the dataset. The if the class label occurrence difference is large then it denotes that there can be a class imbalance problem.

where, TP is true positive, TN is true negative, FP is false positive and FN is false negative[Man22].

5.5 Summary

Using just the available log parses is not plausible due to the different format of data in the OpenStack components. Even though the log parser such as Drain and IPLoM are provide a structural representation of data, they require approximately 1 hour to preprocess 10 million logs also the result they produce cannot be accepted as an input for the models. These parser separates the data in to columns and also gives the number of occurrence but they still need to be parsed further and also needs the preprocessing before it is used as an actual input to the model. Hence forming the regular expression can be used to further parse the logs and the resulting data can be be preprocessed to pass it as an input to the model training. Hence the focus point of research has been kept to suit the needs of OpenStack logs.

Several algorithms are available and have been tested on the OpenStack logs. But in the real world the amount of logs generated by the cloud service provider system is exceptionally high because of the number of datacenter which will have huge number of nodes set up to provide the resources required to customers. Hence the decision to use the unsupervised algorithms have been made. Even thought the supervised algorithms are available and has been tested, they are not suitable to adapt when there is huge number of data being used.

Evaluation metrics along with the expert opinion are the used to evaluate the models. They are also evaluated based on the amount of compute resource and training time required.

6 Design

The DSR requires consists of an important step which involves development of the solution for the defined problem or requirement. Such a development starts with a basic design concept and it improves over a time and results in the systematic solution. This chapter outlines the high-level overview of the "developing step" in DSR in order to understand the entire development of the solution and its strategies involved for the business problem explained in detail in chapter 1 and Chapter 3.

6.1 Steps involved in Design

As explained in DSR methodology, the design of the solution starts with the understanding of the business problem or use case in the environment it is being used as discussed in the Chapter 2. Once we have a concrete understanding it is followed by the design of the solution. In the case of anomaly detection in OpenStack, the solution development starts with the collection of data and preparing the data for the training using ML models, followed by the evaluation of the models. Once the data is preprocessed and ready for training the feature engineering is used to obtain the feature or vectors. This step also involves the dimensionality reduction to reduce the number of attributes generated. These feature dataset is passed as an input to the different models for training and all the models are evaluated based on the expert opinion, evaluation metrics and cost effectiveness. All the steps involved are explained in detail below.

6.1.1 Data Collection

This method involves collection of data from different nodes and different datacenter regions that consists of the information about the OpenStack components such as nova, cinder, keystone, glance and so on. The logs generated by these nodes are huge in number as shown in the figure Figure 4.1. These logs need to be collected from the nodes for training. Due to the data being very large, it is not plausible to determine whether each of the row is a normal log or an anomalous log by the experts. The logs of each component are stored in different folders inside the nodes which would make the extraction of the data easy.

6.1.2 Data Pre-processing and Transformation

This stage involves cleaning and transforming the unstructured data to a structured data in order to feed it as an input for the model training. This step improves the quality of the data which influences the training of the model and the outcome that we shall get after

training. Hence pre-processing is one of the most important steps involved in this solution as logs are highly unstructured and has inconsistent format involving different data types. The data cleaning involves handling missing values, noisy data, solving structural errors. Once the data is cleaned, it must be transformed in order to have the same structure. These steps are very much crucial to avoid the poor performance of the model as many ML models fails to perform well if there are missing values, structural errors or if the data has different format and structure.

6.1.3 Feature Engineering

Feature Engineering refers to the extraction of the attribute, selection of the most suitable feature set for the training and also to reduce the dimensionality of the feature set. The data after preprocessing will be a textual data which needs the feature extraction to be done in order to obtain the feature set that can be passed as an input to the training. In case of real world textual data. the feature set generated can be usually very big and passing such huge number of attributes to the training will affect the model's performance. Hence reducing the number of features while still keeping the important aspects in the dataset would help in training with less computational resources, improve the performance and also avoids over-fitting. The feature engineering is performed after cleaning of the dataset and all the feature values are transformed to have categorical values and scale the data to be in a predefined range. In the next chapter the feature engineering techniques are explained more in detail.

6.1.4 Model Training and Model Selection

The most important step in the final solution decision is the model training. The dataset prepared from the preprocessing and feature engineering is passed as an input to the model for training in order to obtain the classification of the logs. Depending on the performance of the model the hyperparameters are tuned to improve the model's performance. Total of the 6 models are chosen for the comparative study in this study which involve both the classical ML algorithm, most used Deep learning algorithm and the state of the art and novel ML algorithm. The ensemble model involving the combination of the models are experimented in the thesis as well. All the selected models should be trained on a different parameter setting to choose the best fit by making a decision based on its prediction results, efficient use of resources and also the time taken for training. All the implementation details are explained in detail in the next chapter.

6.1.5 Evaluation

After the careful training of the models, they are evaluated based on the quantitative and qualitative assessment along with its resource utilization to determine the best fit. In order to evaluate the model based on the quantitative assessment, evaluation metrics such as precision, recall, f1-score are used. The quality of the data will play a crucial role in the evaluation results. Hence several methods are employed in the thesis to improve the quality of the data for training. The qualitative assessment is done using the

expert opinion on the prediction result by considering whether the number of anomalies predicted are acceptable for the total amount of data used in training and whether the anomalies predicted are correctly classified. Details about the evaluation is explained in Chapter 8.

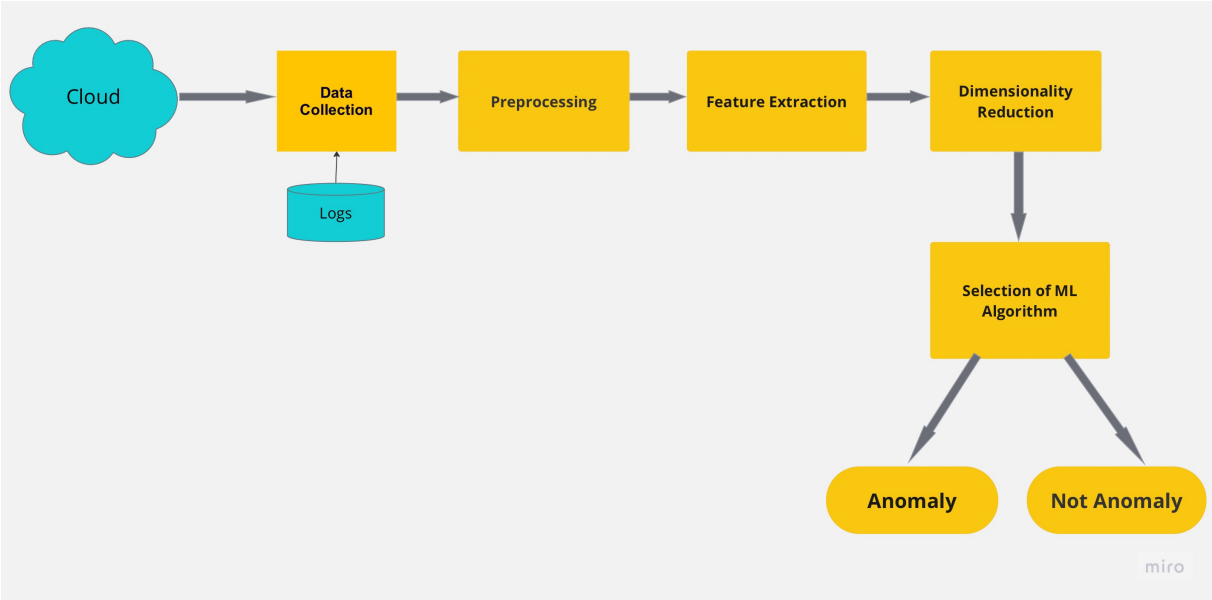


Figure 6.1: Design of the Anomaly Detection

As seen in the above figure Figure 6.1 the final result is to select the best suitable model to use for the anomaly detection. The selected model can be any of the classifiers. In the further chapter the implementation of the design and the results are explained in detail which would give a clear solution to the defined business problem.

7 Implementation

7.1 Data Collection

The dataset used in the thesis is basically logs acquired from the OpenStack components such as Nova, Neutron, Cinder, Keystone, Glance.

SysEleven stack is built upon the OpenStack software which provides the cloud services to 1000 plus customers across European Union.They have 3 data centres located in Berlin and Frankfurt which are named as "DBL", "CBK", "FES". The data is collected from the different OpenStack services that is provided by SysEleven to customers. The logs can be collected for training using three ways, one is to get the logs of each components directly from each node in the datacenter regions as each component's logs are stored in different location. So they can be collected separately. The second way is to get the logs through journald which will not give us the separation of the component logs. The journald also captures other logs as well. Hence a strict filter must be applied to parse the logs to get only the logs relevant to OpenStack components. The third way gives more control and is a feasible option to collect the logs for training. In the SysEleven the logs that are stored in the different nodes are collected through a Fluentd tool and is passed to a Graylog server. Through the Graylog web interface, all the required logs can be exported in to a csv file by using a simple search query such as "nova" for nova related logs.

The collected data is unstructured and consists of several information such as timestamp, verbose level, error message, source, request IDs, HTTP status codes and requests, length and time of the request, ip address, api service information and so on.

The below table Table 7.1 shows the number of logs used for training with respect to each component.

Table 7.1: Number of logs collected

Component	Service	No. of Logs
Nova	Compute	3 million
Neutron	Network	2 million
Cinder	Storage	2 million
Glance	Image	1 million
Keystone	Identity	1 million

7.2 Data Cleaning and Preprocessing

Since the data that will be received from the resources are unstructured, preprocessing such data is a necessary step. There are online resources available such as log parser which uses different algorithms such as Drain, IPLoM to parse the logs and extract the information out of it in to event templates. Just relying on the online parser would result in the loss of information, error in parsing and it takes a lot of time to parse the logs. Hence the regex to suit the log format is chosen and fit to the data which takes about 3 minutes to parse 1 million rows. The online parsers Drain and IPLoM algorithm are selected and tested. The time taken to parse the logs using online parsers are approximately 1 hour for 10 million logs and resulting data was divided under wrong columns if they do not have format starting with timestamp. Further regular expressions were needed to parse the logs correctly which is a tedious task.

The result showed that the segregation is erroneous and hence the custom regex which involved the cleaning of the resulted data that match the log messages are used in order to extract the content required for prediction based on the domain knowledge of OpenStack. The important information such as all the words and http status codes were retrieved after the parsing using the regex.

$$X = (words, http - status - codes) \tag{7.1}$$

All the numbers except status-codes, special characters, hexadecimal characters, URLs are removed from the data and the resulting string is converted to the lower case letters. The resulting data is a plain textual data. The custom stopwords list is added along with the available NLTK stopword list which is available in NLTK library and consists of about 180 English stopwords and removed the negative words since the negative words plays an important role to determine if the cloud is not behaving as expected.

$$custom - list+ = [\prime datacenter - region\prime, \prime daysinaweek\prime] \tag{7.2}$$

$$custom - list- = [\prime negative\ words\ such\ as\ not\ doesn\prime t\ and\ so\ on\prime] \tag{7.3}$$

The below Listing 7.1 and Listing 7.2 shows the resulting data before and after the regex application and stopword removal. After removing the stopwords the data are automatically tokenized.

```
2023-05-10 19:09:47.881 32 INFO nova.metadata.wsgi.server [-] ↩
    10.0.0.93,10.32.144.57,10.32.144.3,127.0.0.1 "GET /latest/meta-data HTTP/1.1" ↩
    status: 200 len: 347 time: 0.0020690
```

Listing 7.1: Sample log before data cleaning

```
[info, nova, metadata, wsgi, server, get, status, 200]
```

Listing 7.2: Sample after data cleaning

7.3 Feature Engineering

The data from the Section 7.2 is passed to the different types of feature engineering techniques. Different feature extraction techniques such as CountVectorizer, Tf-idf, One-hot-Encoding have been used to extract the unique words. All the features generated will be the unique words in the dataset.

7.3.1 CountVectorizer(CV), Tf-idf Vectorizer(Tf-idf), One-hot Encoding

The CV counts the frequency of each word in a sentence and creates a matrix. The CV is implemented using the scikit-learn library with a default hyper parameters. The CV uses the list of sentences as input, hence it is not required to pass the tokenized input. It builds an vocabulary size equal to the unique words in the input.

The Tf-idf vectorizer is also been implemented using the scikit-learn with default hyper parameter setting which results in the document term matrix that consists of number of words in a sentence along with the significance of the word.

The One-hot Encoding has also been implemented using the scikit-learn with the default parameter setting which results in a matrix with the binary values indicating whether the word is present in each sentence or not.

The *fit_transform() and transform()* is used to learn the vocabulary and return the document-term matrix as an output.

The below Figure 7.1 shows the example frequency of 20 features across the dataset.

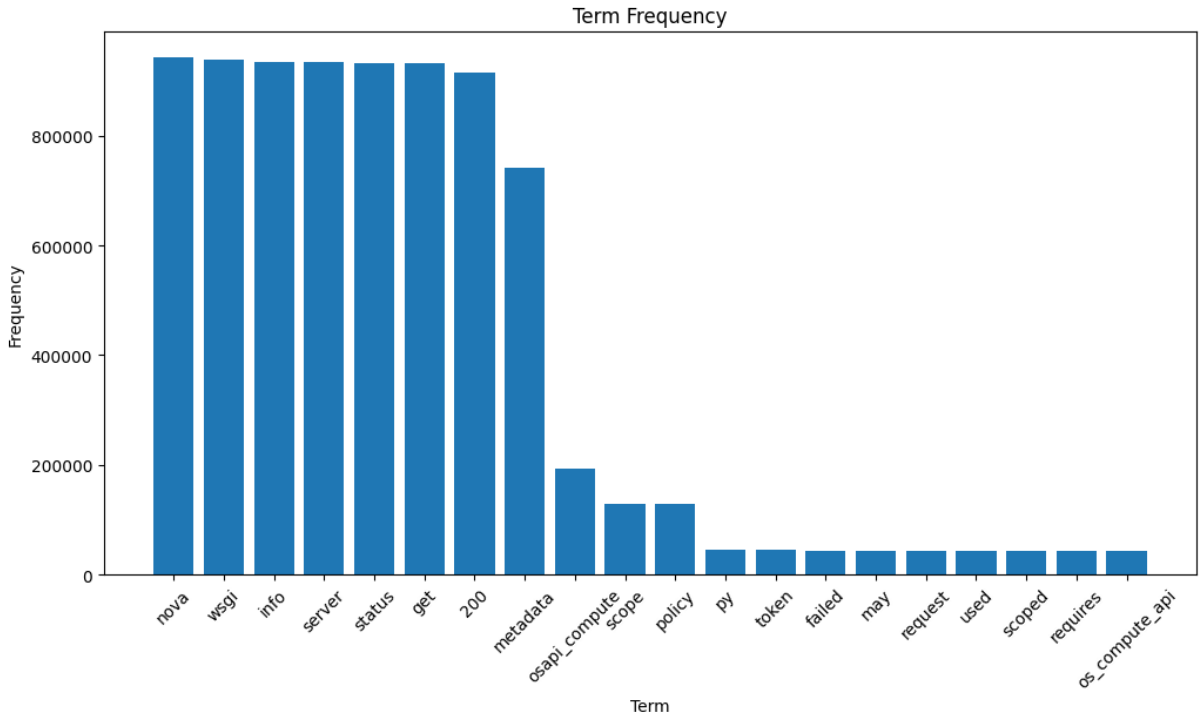


Figure 7.1: Frequency of words across dataset

7.3.2 Word2Vec

The word embeddings are obtained using the Word2vec model which is a NLP technique to get the embeddings. There are several pre trained word2vec models available and in this thesis, the models such as "fasttext-wiki-news-subwords-300" and "word2vec-google-news-300" were used to see if the similarity of the words be derived. But just using these pretrained models will not fit the needs of OpenStack logs as it might not contain some of the exact words such as "wsgi","status-codes", "osapi" and so on. Hence the pre trained word2vec model has been extended to include the vocabulary of OpenStack data. Upon doing this the similarity of the words were available that would fit the data. These values are further reduced using the two dimensionality reduction techniques(PCA and AutoEncoder) and also normalized to get the values between 0 and 1.

7.3.3 Dimensionality Reduction

The features generated are huge in number, hence dimensionality reduction needs to be done. So to reduce the dimension of the features, PCA and AutoEncoder were experimented and compared. The extracted features from the text are passed to the PCA and AutoEncoder to reduce the dimensionlity. The PCA is implemented using scikit-learn and the AutoEncoder is implemented using Tensorflow.

In order to decide the number of componenets for PCA, the cumulative explained variance ratio is plotted and visualized at which point the graph is having the maximum cumulative explained variance. At that point the number of components are chosen. The Figure 7.2 and Figure 7.3 shows the graph of the cumulative explained variance against number of components. According to the graph a suitable number of components is 25 and hence it is selected and tested in thesis.

$$PCA(n\_components = 25) \tag{27}$$

From the image Figure 7.1, the distribution of 20 features across the data points can be seen. Although there are several features that has less occurrence, they cannot be dropped as they might be the perfect candidate for rare case(outlier). For this reason all the features needs to be included and combining those inside the componensts would reduce the dimensionality.

The AutoEncoder are also tested in the thesis for the dimensionality reduction. The bottleneck layer or code size in the AutoEncoder is determined as the number of reduced features. Different code sizes were tested and the setting with minimal loss is selected as the final architecture which has an encoder and decoder layers with 128, 64 neurons and a code size with 32 neurons. The total features were reduced to 32. After training, the reduced feature set is predicted using the hidden layer with predict() function. The activation function ReLU is used in the encoder and decoder layers and the sigmoid is used at the output layer. To compile the model, adam optimizer is used and the loss

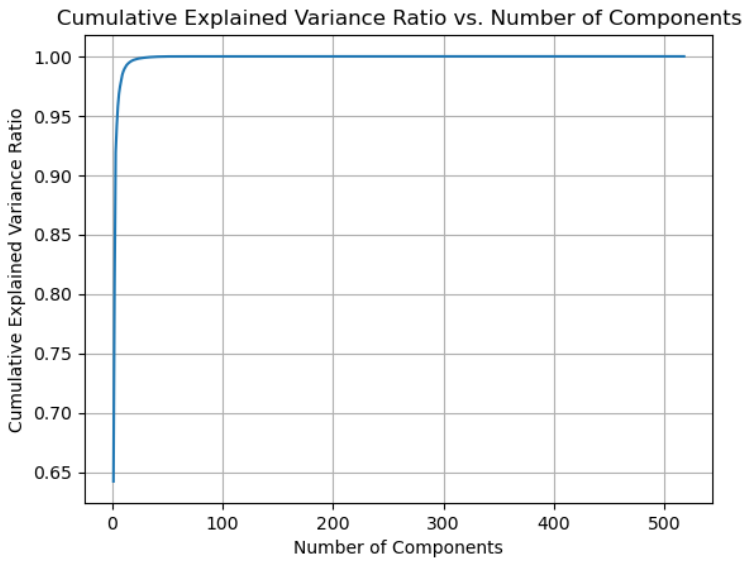


Figure 7.2: Cumulative explained variance ratio

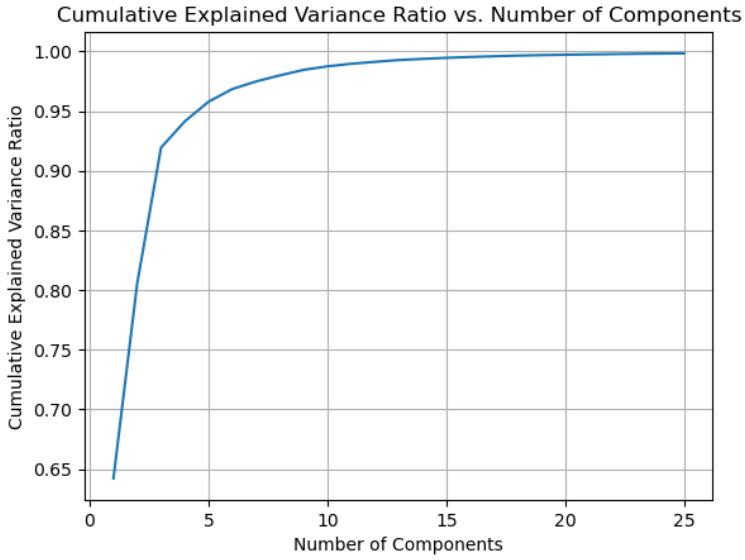


Figure 7.3: Satisfactory n components choice for PCA

function of mean square error is used and it is trained for 10 epochs with a batch size of 128.

After applying feature reduction, we get less features which holds all the features in the combined components.

After the feature extraction and dimensionality reduction the data is normalized where ever applicable using the Min-Max scaler available in scikit-learn so that all the values stay in between 0 and 1 and the data is split randomly in to training and testing. 80% of the data is used for training and 20% is used for testing.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 518)]             0

 dense (Dense)               (None, 128)               66432

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 64)                2112

 dense_4 (Dense)             (None, 128)               8320

 dense_5 (Dense)             (None, 518)               66822

=================================================================
Total params: 154,022
Trainable params: 154,022
Non-trainable params: 0
```

Figure 7.4: AutoEncoder model summary

7.4 Libraries used in the implementation

The OpenStack logs are fed as an input to the classifiers and many in-built libraries are used in the implementation. Libraries used in the implementation is given in the table Table 7.2. The deep learning model such as LSTM and Bi-LSTM are implemented using Tensorflow whereas Iforest is implemented using scikit-learn and COPOD and ECOD are implemented using pyod.

Table 7.2: Libraries used in the implementation

Library	Version
scikit-learn	1.2.2
pyod	1.0.9
tensorflow	2.12.0
pandas	1.5.3
numpy	1.22.4
re	2.2.1

7.4.1 Machine learning Models

As described in the design Chapter 6 all the machine learning models are trained on a different hyper-parameter setting and chosen the best fit for the anomaly detection.

7.4.2 Local Outlier Factor

Local Outlier Factor(LOF) is one of the mostly used traditional algorithm in the research for the unsupervised anomaly detection. The LOF is implemented using the scikit-learn library. The algorithm is implemented using the n_neighbors as 15,20,25 , novelty as true

to use the trained model for unseen data and contamination is selected by incrementing and decrementing it within the range of [0,0.5]. Rest of the settings are used as default for the implementation. Contamination is the percentage of anomalies that are estimated to be present in the dataset.

LocalOutlierFactor(n_neighbors=[15,20,25], contamination=[0,0.5], novelty=True)

The contamination parameter can be selected based on the domain knowledge or by selecting different values to roughly estimate the value based on the model performance. The normalized data is used for training. The fit() function of the constructed model is used to fit the training data. The predict() method is used on the unseen test set and evaluation set to predict the anomalies.

7.4.3 Isolation Forest

This algorithm is also widely used for anomaly detection according to the literature survey and also Isolation forest(IForest) algorithm has been proved to be well suited algorithm for anomaly detection. The algorithm has been tested along with each of the above mentioned list of feature engineering techniques and also have used AutoEncoder and PCA to reduce the dimensionality. The algorithm is tested with different number of estimators and the contamination parameter and with rest of the setting as default. The number of estimators chosen are 200 and contamination is chosen based on the trial and error method and also based on domain knowledge with the values in between [0,0.5].

IsolationForest(n_estimators=200, contamination=[0,0.5])

The fit() function is used to fit the constructed model for training, predict() function is used to predict the anomalies and decision_function() to get the average anomaly score of the dataset.

Isolation Forest with two AutoEncoders for anomaly detection

In [FG20b] an IForest with AutoEncoder is presented which has been experimented in the thesis to check its performance. They claim that IForest with two AutoEncoder gave plausible results. The logs are preprocessed to remove the special characters, numbers, sentences less than 5 words are also dropped. After which the word frequency is computed and the data is normalized and scaled between 0 and 1. The data has been split in to two training sets namely t1 and t2 and a testing set t3. The proportion of the negative and positive logs used are equal. The t1 is passed to the first AutoEncoder for training and the t2 and t3 are passed to the first AutoEncoder to extract features(f1 and f2) by predicting. The first AutoEncoder would reduce the dimensionality and it is done by the code layer. The f1 is fed to the IForest for training and f2 is used to get the prediction and the obtained positive logs(p1) from f2 which are chosen randomly and are trained with second AutoEncoder that has same architecture as the first AutoEncoder. After training, both p1 and p2(the remaining positive and negative logs from IForest output) are fed to the second AutoEncoder to extract features which are denoted as o1 and o2 respectively. The threshold is computed by using the standard deviation(stdv), which is the stdv of the positive predicted data from IForest and a constant c, which is selected based on the

percentage of negative and positive logs in the dataset. The threshold is compared with the mean of each sample in the dataset o2. If the average is less than the threshold then it is considered as anomaly.

This approach has been tested in the thesis by reducing the dimensionality to 32 and with the n_estimators of 200, contamination of 0.015. But with this approach for OpenStack logs, the model performance was poor as it did not detect anomalies because of the threshold selection. The approach also needs the complete knowledge on the logs used for training because it requires equal number of negative and positive logs. Because of these reasons this approach did not fit the use case.

7.4.4 COPOD

As per the literature survey the COPOD is a novel-outlier detection algorithm which has not been used for the log-based anomaly detection to the best of our knowledge. Existing approaches have high computational complexity and hence COPOD can be used as it is parameter-free. This algorithm uses Empirical Cumulative Distribution Functions(ECDFs) and there is no need of hyper parameter tuning as well. The only setting that needs to be taken care is contamination of the dataset. The contamination is estimated to be between [0,0.5].

COPOD(contamination=[0,0.5])

The constructed model is used with fit() function for training and predict() function for predicting the anomalies. There are several attributes available which is used for the evaluation of the model performance. The decision_score() function is used to get the scores of the outliers in the training data. The higher score indicate the sentence being abnormal. The threshold is obtained based on the contamination and the number of observations. The output generated is a binary label which indicates 0 being an inliner and 1 being an outlier. The fitted model can also be used to get raw anomaly scores by using the decision_function().

7.4.5 ECOD

It is an algorithm that also uses Empirical Cumulative Distribution Functions(ECDFs) and is also a parameter free and this algorithm allows the understanding of the reasoning behind the prediction. The only setting that needs to be taken care here is the contamination parameter which can be selected between [0,0.5] based on the domain knowledge and also based on the experiment with the different contamination values.

ECOD(contamination=[0,0.5])

The fit() function is used on the constructed model and the predict() function is used to predict the outliers in the unseen data.Similar to COPOD, the ECOD implementation also give the attributes for the better understanding and evaluation of the model performance. The decision-function(), decision-score() are used to get the anomaly scores by using fitted model and the outlier scores. The output is also a binary, indication 1 being an outlier and 0 being an inliner.

7.4.6 LSTM and Bi-LSTM

To analyze the texts in the sequential way and to detect the anomalies, LSTMs and Bi-LSTMs can be used. The LSTMs are primarily used for supervised learning. But in order to use LSTMs for the unsupervised learning, they need to be combined with AutoEncoder. The dataset is preprocessed before feeding it in to the model as input. The preprocessed dataset are transformed in to the sequence of vectors and are normalized. The model is constructed using Keras library. The model is a Encoder-Decoder LSTM architecture which is built as sequential meaning each layer is built on top of the other and needs different hyper parameter settings. The overall architecture of the model is explained below.

1. The shape of the input is determined based on the dimensions of the input data.

2.The encoder-layer 1 is the LSTM with 128 units. The activation parameter is ReLU and the kernel_regularizer applies L2 regularization to the layer's weights. The encoder-layer 2 is the LSTM layer with 64 units and the return_sequences=False meaning that this layer does not return sequences but rather a single output vector.

3. The third layer is a RepeatVector and it replicates the output in order to match the shape of the input.

4. The fourth layer is decoder-layer 1 which is a LSTM of 64 units and activation function ReLU which takes the repeated output from the previous layer as input.

5. The fifth layer is the decoder-layer 2 which is a LSTM of 128 units and activation function ReLU.

6.The last layer is the TimeDistributed which is a Dense layer to each time step of the output sequence from the fourth LSTM layer. The TimeDistributed wrapper allows the dense layer to be applied independently at each time step. The number of units in the dense layer matches the number of features in the input data.

model = Model(inputs=inputs, outputs=output): This line defines the complete AutoEncoder model with the input and output layers specified.

Overall, this architecture consists of multiple LSTM layers to encode and decode the input data. The intermediate layers gradually reduce the dimensionality of the input, and the final output layer aims to reconstruct the input sequence.

The constructed neural network is compiled using the loss function as mean squared error, number of epoch is 30 with batch size of 128 and an optimizer Adam.

Bi-LSTM

Since the LSTM handles the data flow in a unidirectional fashion, Bi-LSTM is implemented in the thesis to handle the bi-directional data flow. The Bi-LSTM process the data in both the directions. The architecture of Bi-LSTM is similar to the LSTM and is implemented using the wrapping of Bidirectional package from the Keras around LSTM layers.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 1, 518)]          0

lstm_20 (LSTM)               (None, 1, 128)            331264

lstm_21 (LSTM)               (None, 64)                49408

repeat_vector_5 (RepeatVect  (None, 1, 64)             0
or)

lstm_22 (LSTM)               (None, 1, 64)             33024

lstm_23 (LSTM)               (None, 1, 128)            98816

time_distributed_5 (TimeDis  (None, 1, 518)            66822
tributed)

=================================================================
Total params: 579,334
Trainable params: 579,334
Non-trainable params: 0
```

Figure 7.5: LSTM model summary

The above layer builds the hidden layer of an LSTM in two copies. One is to fit the sequence exactly as it is and the other will build the input sequence backwards. The hyperparameters used are same values as LSTM. The model summary of the Bi-LSTM is given below.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 1, 518)]          0

bidirectional (Bidirectiona  (None, 1, 256)            662528
l)

bidirectional_1 (Bidirectio  (None, 128)               164352
nal)

repeat_vector_2 (RepeatVect  (None, 1, 128)            0
or)

bidirectional_2 (Bidirectio  (None, 1, 128)            98816
nal)

bidirectional_3 (Bidirectio  (None, 1, 256)            263168
nal)

time_distributed_2 (TimeDis  (None, 1, 518)            133126
tributed)
```

Figure 7.6: Bi-LSTM model summary

7.4.7 Ensemble models

The ensemble model is the custom model of the multiple algorithms used for the training. The ensemble models consists of different combination of models with different hyperparameter settings that can be used to train the model which will combine the capabilities of all the models used in to one model. After obtaining the results of all the individual models the best performing models were selected to combine and build it as an Ensemble model. Two such models were built in order to combine their predic-

tion results and check if the combination of models can perform better in detecting the anomalies.

The Ensemble1 has the combination of IForest, COPOD, ECOD and LSTM whereas the Ensemble2 has the combination of IForest, COPOD, ECOD.

The model is constructed in the same way as the original implementation of the listed algorithms and they are appended to the list and used for training. The hyper parameters selected are already tested in the individual implementation of the model and the contamination is selected between [0,0.5]. The constructed models use fit() function to fit the training data and predict() function is used to predict each models output. The final prediction is determined by voting where the maximum of votes is taken to decide the final prediction. In order to have a tie breaker in case of even stack such as Ensemble1, a final prediction is determined by implementing a tie breaking rule to choose a random choice. The resulting values obtained are -1 and 1, where -1 indicates anomalies and 1 indicated not an anomaly.

7.4.8 Deployment

The built solution is deployed on a server in SysEleven overlay infrastructure as a prototype. The first step in the pipeline is to collect the logs from all the nodes in all the datacenters and pass to the fluentd[1] which is an opensource data collector. This act as bridge between the nodes and the monitoring server that runs the machine learning prediction model.

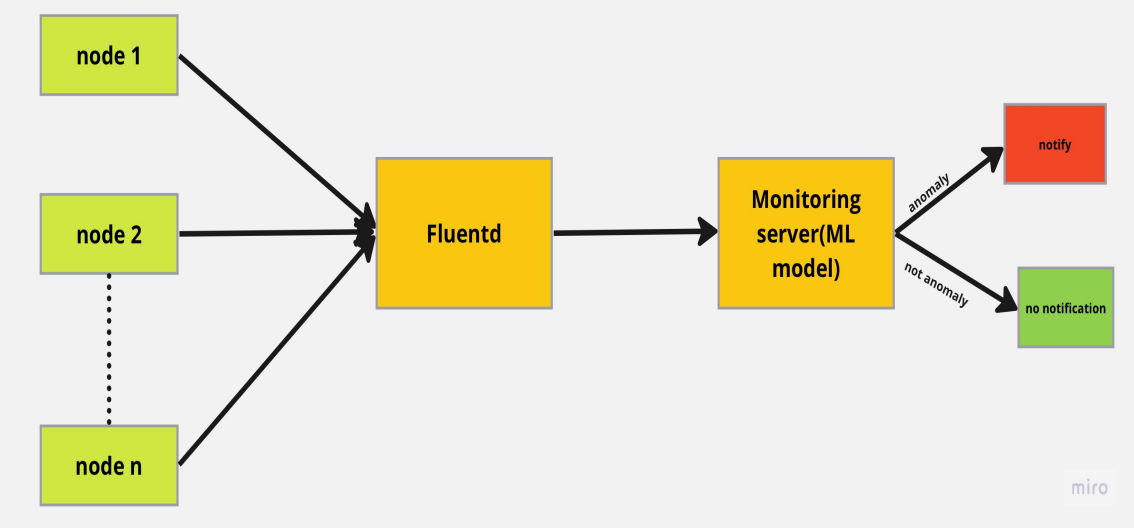


Figure 7.7: Deployment of ML model for Notification

Fluentd provides several plugins that can be defined according to the needs. The fluentd collects the logs by defining the input plugin and here <match> plugin can be used to define the required logs that match the regular expression to be used as input. These logs are forwarded to the monitoring server which uses the port number and host as input

[1]Fluentd is a unified data collector which is used for the collection of data in order to understand it better[Flu]

in the config file. These logs can be stored in the location specified. Then our model can take these logs, preprocess and predict the anomalies. Once the prediction result is obtained, if it is anomaly then it is notified via email using SMTP(simple mail transfer protocol).

The code for the entire implementation has been uploaded in the github[2].

[2]https://github.com/Deeksha282528/Master-thesis

8 Results and Evaluation

The evaluation is a very crucial step in DSR, which determines the quality of the solution developed and if the solution built addresses the business requirements. The design of the evaluation has been already discussed in the previous section. This chapter focuses on the experimental results and evaluation measures that are used to evaluate the models performance against each other.

8.1 Experimental Setup

The data cleaning, feature engineering techniques, models are developed using the python 3.8.10 using Jupyter notebook and SysEleven environment. The system specification used in the experiment is given below.

Table 8.1: System Specifications For Experiment

System Component	Specification
CPU	Intel Core Processor (Broadwell, IBRS) @ 2.39GHz
RAM	64 GiB
Disk	800GiB
CPU Optimised-API name in SysEleven Stack	l1c.2xlarge

8.2 Result

The experimental results of the models used are discussed in this section for the anomaly detection. The experiments include different settings for the models and also the comparative analysis of the results for the model is also discussed in this section.

As described in the literature survey, the models are selected based on its frequency of usage for this business problem and whether the model is the state of the art and novel. The LOF, LSTM/Bi-LSTM and Iforest are the most used ones and also are the state of the art for the business problem and for the OpenStack logs. The COPOD and ECOD are the novel algorithms that are claimed to perform well on the large datasets with less computational efforts as per the literature survey and they are never been tested for OpenStack anomaly detection to the best of our knowledge.

Each component of OpenStack is trained separately and the dataset of each component is divided in to train and test dataset which uses 80% for training and 20% for testing. In order to evaluate the trained model the additional data has been used that had manually induced anomalies, anomalies collected from the logs and also the anomalies verified by the expert interview. This is used as a ground truth dataset for the evaluation of the models. This unseen dataset consists of 102000 samples among which 3737(3.66%) are anomalies that are manually added and rest are not an anomaly. In the further explanation the normal data is also known as positive data and the anomalous data is also referred as negative data.

8.2.1 Local Outlier Factor

The LOF is experimented with the parameter setting of n_neighbors=15 ,the contamination value is set to 0.065 and novelty is set to true in order to predict on the new unseen data during training. This is trained on the training dataset of each component and is validated using the test set.

The classification report for the LOF prediction with feature extraction technique using CountVectorizer has been trained using all the features extracted from the CountVectorizer method and the result has been given below in Table 8.2. Even though the predicted anomalies were all true anomalies, the model performance is poor in detecting all the anomalies. The model performed well in predicting the positive samples.

Table 8.2: Classification Report for LOF with CountVectorizer

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.54	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

The classification report for LOF with CountVectorizer and PCA is shown in the Table 8.3. The performance of the model did improve in predicting the negative data. But it failed to reduce the number of false positives.

Table 8.3: Classification Report for LOF with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.57	0.83	0.67	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.98	102000
Macro avg	0.78	0.90	0.83	102000
weighted avg	0.98	0.96	0.97	102000

In Table 8.4, the model prediction is very poor. It predicted all the data as anomalies. The contamination was adjusted by performing grid search which showed *n_neighbors* to be 5 and contamination to be 0.001. Even though the parameters were increased, the result did not improve.

Table 8.4: Classification Report for LOF with CountVectorizer and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.04	1.00	0.07	3737
Not Anomaly	0.00	0.00	0.00	98263
Accuracy			0.04	102000
Macro avg	0.02	0.50	0.04	102000
weighted avg	0.00	0.04	0.00	102000

In the Table 8.5, the classification report for LOF with One-Hot encoding is used. The LOF performed very poorly in this case. It just predicted 3.8%(143) anomalies out of the 3737 with a contamination of 0.065. Even increasing the contamination did not improve the results and the model got over-fitted by increasing the contamination.

Table 8.5: Classification Report for LOF with One-Hot

	Precision	Recall	F1-Score	Support
Anomaly	0.98	0.04	0.07	3737
Not Anomaly	0.96	1.00	0.98	98263
Accuracy			0.96	102000
Macro avg	0.97	0.52	0.53	102000
weighted avg	0.97	0.96	0.95	102000

With the PCA dimensionality reduction the LOF model's performance improved where it detected the 94% of anomalies.

Table 8.6: Classification Report for LOF with One-Hot and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.30	0.94	0.46	3737
Not Anomaly	1.00	0.92	0.98	98263
Accuracy			0.92	102000
Macro avg	0.65	0.93	0.71	102000
weighted avg	0.97	0.92	0.94	102000

In Table 8.7, the model predicted all the samples in the unseen validation set as anomalies. But the false positive rate is too high to say that the model performed well.

In Table 8.8, the model prediction is very poor because it predicted all the data as anomalies. The contamination was adjusted and even after increasing and decreasing the

Table 8.7: Classification Report for LOF with One-Hot and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.04	1.00	0.07	3737
Not Anomaly	0.00	0.00	0.00	98263
Accuracy			0.04	102000
Macro avg	0.02	0.50	0.04	102000
weighted avg	0.00	0.04	0.00	102000

contamination, the result did not improve. The performance was also poor for Tf-idf with PCA and AutoEncoder used as dimensionality reduction because of high false positive rate and the result is shown in Table 8.9 and Table 8.10.

Table 8.8: Classification Report for LOF with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.25	0.45	0.32	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.62	0.70	0.64	102000
weighted avg	0.95	0.93	0.94	102000

Table 8.9: Classification Report for LOF with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.04	1.00	0.07	3737
Not Anomaly	0.00	0.00	0.00	98263
Accuracy			0.04	102000
Macro avg	0.02	0.50	0.04	102000
weighted avg	0.00	0.04	0.00	102000

The results of Word2Vec is shown in Table 8.11, Table 8.12 and Table 8.13. Only 4% of anomalies were captured in case of using just Word2Vec, but almost all of them are anomalies. Hence the precision score is 98%. But this model generated high number of false positives. Even the usage of PCA did not improve the results.

8.2.2 Isolation Forest

The Iforest model is built with the parameter setting of *n_estimators* and defining the percentage of contamination. The same test data that has unseen samples of anomalies are used for prediction in the Iforest. In the Table 8.14, the classification report for the Iforest model with countvectorizer is shown. Although the model performed well in predicting the positive data, its performance is poor in predicting the anomalous data. The model predicted more number of outliers which involved false positives for a contamination of

Table 8.10: Classification Report for LOF with Tf-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.04	1.00	0.07	3737
Not Anomaly	0.00	0.00	0.00	98263
Accuracy			0.04	102000
Macro avg	0.02	0.50	0.04	102000
weighted avg	0.00	0.04	0.00	102000

Table 8.11: Classification Report for LOF with Word2Vec

	Precision	Recall	F1-Score	Support
Anomaly	0.98	0.04	0.07	3737
Not Anomaly	0.96	1.00	0.98	98263
Accuracy			0.96	102000
Macro avg	0.97	0.52	0.53	102000
weighted avg	0.97	0.96	0.95	102000

Table 8.12: Classification Report for LOF with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.98	0.04	0.07	3737
Not Anomaly	0.96	1.00	0.98	98263
Accuracy			0.96	102000
Macro avg	0.97	0.52	0.53	102000
weighted avg	0.97	0.96	0.95	102000

Table 8.13: Classification Report for LOF with Word2Vec and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.98	0.03	0.07	3737
Not Anomaly	0.96	1.00	0.98	98263
Accuracy			0.96	102000
Macro avg	0.97	0.52	0.52	102000
weighted avg	0.97	0.96	0.95	102000

0.065, because of which the precision value is just 38%. But increasing this number resulted in high number of false negative. The training time required to train the model was approximately 2 hours for 1 million rows.

Table 8.14: Classification Report for Iforest and CV

	Precision	Recall	F1-Score	Support
Anomaly	0.38	0.82	0.52	3737
Not Anomaly	0.99	0.95	0.97	98263
Accuracy			0.94	102000
Macro avg	0.69	0.88	0.75	102000
weighted avg	0.97	0.94	0.95	102000

The table Table 8.15 shows the result of Iforest with CountVectorizer for feature extraction and PCA for dimensionality reduction. Using PCA to reduce the dimensions to capture all the information or with minimal loss of information within 25 components resulted better in capturing the anomalies. The training took about 42 minutes to train the 1 million rows.

Table 8.15: Classification Report for Iforest with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.83	0.96	0.89	3737
Not Anomaly	1.00	0.99	1.00	98263
Accuracy			0.99	102000
Macro avg	0.92	0.98	0.95	102000
weighted avg	0.99	0.99	0.99	102000

In Table 8.16, the Iforest is trained with CountVectorizer and AutoEncoder with 32 features to detect anomalies. These features are obtained by the code layer. The model's performance is poor in detecting anomalies and its recall is low in predicting positive samples when compared to other two models.

Table 8.16: Classification Report for Iforest with AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.38	0.82	0.52	3737
Not Anomaly	0.99	0.95	0.97	98263
Accuracy			0.94	102000
Macro avg	0.69	0.88	0.75	102000
weighted avg	0.97	0.94	0.95	102000

The table below shows the results for Table 8.17. Among the predicted anomalies, 57% are true anomalies and model is able to predict 80% of the anomalies correctly. The false positive rate is higher in this case as well. The contamination used is 0.065.

Table 8.17: Classification Report for Iforest with onehot

	Precision	Recall	F1-Score	Support
Anomaly	0.57	0.80	0.67	3737
Not Anomaly	0.99	0.98	0.98	98263
Accuracy			0.97	102000
Macro avg	0.78	0.89	0.83	102000
weighted avg	0.98	0.97	0.97	102000

In Table 8.18 the One-Hot encoding is used along with the PCA which is used as a dimensionality reduction and it resulted in 50% precision and all the anomalies were captured. But false positive rate is high in this case.

Table 8.18: Classification Report for Iforest with onehot and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.59	1.00	0.74	3737
Not Anomaly	1.00	0.97	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.99	0.86	102000
weighted avg	0.98	0.97	0.98	102000

The Table 8.19 shows the classification report of Iforest with One-Hot Encoding and AutoEncoder for dimensionality reduction. The model performed better in comparison with the one-hot encoding and one-hot encoding along with PCA with precision of 73% and recall 87%.

Table 8.19: Classification Report for Iforest with onehot and AE

	Precision	Recall	F1-Score	Support
Anomaly	0.73	0.87	0.80	3737
Not Anomaly	1.00	0.99	0.99	98263
Accuracy			0.98	102000
Macro avg	0.86	0.93	0.89	102000
weighted avg	0.98	0.98	0.98	102000

The Iforest model is experimented with the Tf-idf, Tf-idf along with PCA and Tf-idf along with AutoEncoders and the results of the model is shown in the Table 8.20, Table 8.21, Table 8.22 respectively. All though the model performed well in predicting positive data, its performance is very poor in predicting the anomalies. The Tf-idf along with PCA was able to predict all the outliers/anomalies, it has high number of false positive prediction.

The Iforest with Word2Vec is used to train the word embeddings. The vocabulary is extended by adding the words from the OpenStack dataset. The dimensionality reduction

Table 8.20: Classification Report for Iforest with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.25	0.45	0.32	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.62	0.70	0.64	102000
weighted avg	0.95	0.93	0.94	102000

Table 8.21: Classification Report for Iforest with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.58	1.00	0.74	3737
Not Anomaly	1.00	0.97	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.99	0.86	102000
weighted avg	0.98	0.97	0.98	102000

Table 8.22: Classification Report for Iforest with Tf-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.45	0.56	0.50	3737
Not Anomaly	0.98	0.97	0.98	98263
Accuracy			0.96	102000
Macro avg	0.72	0.77	0.74	102000
weighted avg	0.96	0.96	0.96	102000

using PCA for the word2vec performed well as the precision and recall are higher when compared to simply using all the features or using the dimensionality reduction by AutoEncoder. The result of Word2Vec with Iforest along with the PCA and AutoEncoders are shown in Table 8.23, Table 8.24, Table 8.25.

Table 8.23: Classification Report for Iforest with W2vec

	Precision	Recall	F1-Score	Support
Anomaly	0.65	0.92	0.76	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.98	102000
Macro avg	0.83	0.95	0.88	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.24: Classification Report for Iforest with W2vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.82	0.97	0.89	3737
Not Anomaly	1.00	0.99	1.00	98263
Accuracy			0.99	102000
Macro avg	0.91	0.98	0.94	102000
weighted avg	0.99	0.99	0.99	102000

Table 8.25: Classification Report for Iforest with W2vec and AE

	Precision	Recall	F1-Score	Support
Anomaly	0.59	0.91	0.71	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.94	0.85	102000
weighted avg	0.98	0.97	0.98	102000

8.2.3 COPOD

This section explains about the results obtained using the COPOD for classification. The model is tested against the same cases as of the above two algorithms and the result of each case is shown below in the table. This algorithm does not require hyper parameter tuning but it expects the contamination percentage to be defined.

The below Table 8.26 shows the classification report for the COPOD with CountVectorizer that extracts the features from the text data. The CountVectorizer with PCA performed well in comparison with just CountVectorizer and CountVectorizer with AutoEncoder as shown in the Table 8.27 and Table 8.28.

Table 8.26: Classification Report for COPOD with CountVectorizer

	Precision	Recall	F1-Score	Support
Anomaly	0.29	0.56	0.38	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.64	0.75	0.67	102000
weighted avg	0.96	0.93	0.94	102000

Table 8.27: Classification Report for COPOD with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.71	1.00	0.83	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.99	102000
Macro avg	0.86	0.99	0.91	102000
weighted avg	0.99	0.99	0.99	102000

Table 8.28: Classification Report for COPOD with CountVectorizer and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.11	0.17	0.13	3737
Not Anomaly	0.97	0.95	0.96	98263
Accuracy			0.92	102000
Macro avg	0.54	0.56	0.55	102000
weighted avg	0.94	0.92	0.93	102000

In case of COPOD with One-hot encoding the AutoEncoder dimensionality reduction performed very poorly. The PCA technique was able to capture all the anomalies but the false positive rate in more than 50%. the results are shown in Table 8.29, Table 8.30 and Table 8.31. The model with AutoEncoders performance is very poor in comaparison to the other two feature engineering methods.

Table 8.29: Classification Report for COPOD with One-Hot

	Precision	Recall	F1-Score	Support
Anomaly	0.29	0.56	0.38	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.64	0.75	0.67	102000
weighted avg	0.96	0.93	0.94	102000

Table 8.30: Classification Report for COPOD with One-Hot and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.41	1.00	0.58	3737
Not Anomaly	1.00	0.95	0.97	98263
Accuracy			0.95	102000
Macro avg	0.70	0.97	0.78	102000
weighted avg	0.98	0.95	0.96	102000

Table 8.31: Classification Report for COPOD with One-Hot and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.05	0.06	0.05	3737
Not Anomaly	0.96	0.95	0.96	98263
Accuracy			0.92	102000
Macro avg	0.51	0.51	0.51	102000
weighted avg	0.93	0.92	0.93	102000

In case of using the Tf-idf with COPOD, only 54% of the anomalies were captured resulting in large number false positives. The Tf-idf with PCA have a better balance between precision and recall, as it predicted 82% correct prediction out of the total anomaly predicted. In case of COPOD with Tf-idf and AutoEncoder, all the anomalies were captured but there is a high number of false positives. The results for the Tf-idf is shown in Table 8.32, Table 8.33, Table 8.34.

The results for COPOD with Word2Vec is given in the Table 8.35, Table 8.36 and Table 8.37. All 3 variants could capture only 62% of the anomalies and COPOD with just the Word2Vec predicted less number of false positives.

Table 8.32: Classification Report for COPOD with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.31	0.54	0.40	3737
Not Anomaly	0.98	0.96	0.97	98263
Accuracy			0.94	102000
Macro avg	0.65	0.75	0.68	102000
weighted avg	0.96	0.94	0.95	102000

Table 8.33: Classification Report for COPOD with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	**0.82**	**1.00**	**0.90**	3737
Not Anomaly	1.00	0.99	1.00	98263
Accuracy			0.99	102000
Macro avg	0.91	1.00	0.95	102000
weighted avg	0.99	0.99	0.99	102000

Table 8.34: Classification Report for COPOD with Tf-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.41	1.00	0.58	3737
Not Anomaly	1.00	0.95	0.97	98263
Accuracy			0.95	102000
Macro avg	0.70	0.97	0.78	102000
weighted avg	0.98	0.95	0.96	102000

Table 8.35: Classification Report for COPOD with Word2Vec

	Precision	Recall	F1-Score	Support
Anomaly	0.30	0.62	0.40	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.64	0.78	0.68	102000
weighted avg	0.96	0.93	0.94	102000

Table 8.36: Classification Report for COPOD with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.47	0.62	0.53	3737
Not Anomaly	0.99	0.97	0.98	98263
Accuracy			0.96	102000
Macro avg	0.73	0.79	0.76	102000
weighted avg	0.97	0.96	0.96	102000

Table 8.37: Classification Report for COPOD with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.48	0.62	0.54	3737
Not Anomaly	0.99	0.97	0.98	98263
Accuracy			0.96	102000
Macro avg	0.73	0.80	0.76	102000
weighted avg	0.97	0.96	0.96	102000

8.2.4 ECOD

ECOD uses emperical cumulative distribution factor to detect the anomalies/outliers. This algorithm is also parameter free and needs to specify the contamination value. The results are computed for same cases as shown. The results with CountVectorizer is shown in Table 8.38, Table 8.39 and Table 8.40. The performance of CountVectorizer with AutoEncoder improved and 99% of the anomalies were captured but false positive rate is higher. The CountVectorizer with PCA captured all the anomalies but also about 30% of the anomalies predicted were false positives.

Table 8.38: Classification Report for ECOD with CountVectorizer

	Precision	Recall	F1-Score	Support
Anomaly	0.29	0.56	0.38	3737
Not Anomaly	0.98	0.95	0.96	98263
Accuracy			0.93	102000
Macro avg	0.64	0.75	0.67	102000
weighted avg	0.96	0.93	0.94	102000

Table 8.39: Classification Report for ECOD with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	**0.72**	**1.00**	**0.83**	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.99	102000
Macro avg	0.86	0.99	0.91	102000
weighted avg	0.99	0.99	0.99	102000

The one-hot encoding results are shown in the Table 8.41, Table 8.42 and Table 8.43. It shows that the rate of false positives are very high and the model with PCA was able to capture all the anomalies.

The results are shown in Table 8.44, Table 8.45 and Table 8.46 for the ECOD with Tf-idf. The Tf-idf with PCA and the Tf-if with the AutoEncoder predicted all the anomalies. But the false positive rate is higher in Tf-idf with AutoEncoder when compared to the PCA.

Table 8.40: Classification Report for ECOD with CountVectorizer and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.41	0.99	0.58	3737
Not Anomaly	1.00	0.95	0.97	98263
Accuracy			0.95	102000
Macro avg	0.70	0.97	0.78	102000
weighted avg	0.98	0.95	0.96	102000

Table 8.41: Classification Report for ECOD with One-Hot

	Precision	Recall	F1-Score	Support
Anomaly	0.55	0.54	0.54	3737
Not Anomaly	0.98	0.98	0.98	98263
Accuracy			0.97	102000
Macro avg	0.76	0.76	0.76	102000
weighted avg	0.97	0.97	0.97	102000

Table 8.42: Classification Report for ECOD with One-Hot and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.57	1.00	0.73	3737
Not Anomaly	1.00	0.97	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.99	0.86	102000
weighted avg	0.98	0.97	0.98	102000

Table 8.43: Classification Report for ECOD with One-Hot and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.05	0.06	0.05	3737
Not Anomaly	0.96	0.95	0.96	98263
Accuracy			0.92	102000
Macro avg	0.51	0.51	0.51	102000
weighted avg	0.93	0.92	0.93	102000

Table 8.44: Classification Report for ECOD with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.29	0.56	0.38	3737
Not Anomaly	0.98	0.95	0.97	98263
Accuracy			0.93	102000
Macro avg	0.64	0.75	0.67	102000
weighted avg	0.96	0.93	0.94	102000

Table 8.45: Classification Report for ECOD with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.65	1.00	0.79	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.98	102000
Macro avg	0.82	0.99	0.89	102000
weighted avg	0.99	0.98	0.98	102000

Table 8.46: Classification Report for ECOD with Tf-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.41	1.00	0.58	3737
Not Anomaly	1.00	0.95	0.97	98263
Accuracy			0.95	102000
Macro avg	0.70	0.97	0.78	102000
weighted avg	0.98	0.95	0.96	102000

The ECOD with word2vec failed to predict all the anomalies even with the feature reduction. The performance with PCA was able to capture 62% of the total anomalies. This report is shown in Table 8.47, Table 8.48 and Table 8.49.

Table 8.47: Classification Report for ECOD with Word2Vec

	Precision	Recall	F1-Score	Support
Anomaly	0.47	0.62	0.53	3737
Not Anomaly	0.99	0.97	0.98	98263
Accuracy			0.96	102000
Macro avg	0.73	0.79	0.76	102000
weighted avg	0.98	0.95	0.96	102000

8.2.5 LSTM/Bi-LSTM

The LSTM is the deep learning model which has been used more in case of anomaly detection in cloud. Even though the LSTM is primarily used for supervised learning, this can be adapted for unsupervised learning by using AutoEncoder. In the thesis, the experiments are conducted using both LSTM and Bi-LSTM as the LSTM only remembers the previous data, but Bi-LSTM remembers both the previous and the next data of the target. The experiment required hyper parameter tuning and also the threshold to be defined based on the distribution of the loss for the data.

The results of LSTM and Bi-LSTM used for Countvectorizer is given in the Table 8.50, Table 8.51 and Table 8.52. The LSTM with Countvectorizer and PCA performed really well in capturing the anomalous data. The usage of AutoEncoder for dimensionality

Table 8.48: Classification Report for ECOD with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.85	0.62	0.71	3737
Not Anomaly	0.99	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.92	0.81	0.85	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.49: Classification Report for ECOD with Word2Vec and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.58	0.54	0.56	3737
Not Anomaly	0.98	0.99	0.98	98263
Accuracy			0.97	102000
Macro avg	0.78	0.76	0.77	102000
weighted avg	0.97	0.97	0.97	102000

reduction with LSTM improved when compared to other models. But the percentage of prediction of anomaly is very low.

Table 8.50: Classification Report for LSTM with CountVectorizer

	Precision	Recall	F1-Score	Support
Anomaly	0.99	0.54	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

The result report in Table 8.53, Table 8.54 and Table 8.55 shows the result of the One-hot encoding being used with LSTM. The LSTM with just one-hot encoding was able to predict all the anomalies, but the false positive rate is higher. The other two methods were able to capture approximately 50% of the anomalies.

The TF-idf with AutoEncoder as a dimensionality reduction performed really well by capturing all the anomalous data with 64% of the data predicted ass anomalies are actually true and this 64% covers the entire rows of anomalous data in the test set. The result is shown in Table 8.56, Table 8.57 and Table 8.58.

The LSTM performance with word2vec is shown in the Table 8.59, Table 8.60 and Table 8.61. The model with PCA captured all the anomalous data but the false positive rate is more than 50%. The AutoEncoder perfomance has also seem to better when compared to the trend in other models.

The Bi-LSTM is used by creating a wrapper of Bidirectional layer around the LSTM

Table 8.51: Classification Report for LSTM with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	**0.80**	**0.93**	**0.86**	3737
Not Anomaly	1.00	0.99	0.99	98263
Accuracy			0.99	102000
Macro avg	0.87	0.98	0.92	102000
weighted avg	0.99	0.99	0.99	102000

Table 8.52: Classification Report for LSTM with CountVectorizer and Autoebcoder

	Precision	Recall	F1-Score	Support
Anomaly	0.61	0.62	0.62	3737
Not Anomaly	0.99	0.99	0.99	98263
Accuracy			0.97	102000
Macro avg	0.80	0.80	0.80	102000
weighted avg	0.97	0.97	0.97	102000

Table 8.53: Classification Report for LSTM with One-Hot Encoding

	Precision	Recall	F1-Score	Support
Anomaly	0.44	1.00	0.61	3737
Not Anomaly	1.00	0.95	0.98	98263
Accuracy			0.95	102000
Macro avg	0.72	0.98	0.79	102000
weighted avg	0.98	0.95	0.96	102000

Table 8.54: Classification Report for LSTM with One-Hot encoding and PCA

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.53	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.55: Classification Report for LSTM with One-Hot encoding and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.90	0.58	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.94	0.79	0.85	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.56: Classification Report for LSTM with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.92	0.56	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.95	0.78	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.57: Classification Report for LSTM with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.54	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.58: Classification Report for LSTM with TF-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.64	1.00	0.78	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.98	102000
Macro avg	0.82	0.99	0.88	102000
weighted avg	0.99	0.98	0.98	102000

Table 8.59: Classification Report for LSTM with Word2Vec

	Precision	Recall	F1-Score	Support
Anomaly	0.99	0.44	0.61	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.98	0.72	0.80	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.60: Classification Report for LSTM with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.58	1.00	0.74	3737
Not Anomaly	1.00	0.97	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.0.99	0.86	102000
weighted avg	0.98	0.97	0.98	102000

Table 8.61: Classification Report for LSTM with Word2Vec and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.61	0.58	0.59	3737
Not Anomaly	0.98	0.99	0.98	98263
Accuracy			0.97	102000
Macro avg	0.79	0.0.78	0.79	102000
weighted avg	0.97	0.97	0.97	102000

layers in the AutoEncoder. The same techniques of feature engineering is used and the results are shown.

The Countvectorizer with AutoEncoder performed well in comparison with the other two cases. But the number of anomalies captured and the false positive rate is very high. The CountVectorizer results are shown in Table 8.62, Table 8.63 and Table 8.64

Table 8.62: Classification Report for Bi-LSTM with CountVectorizer

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.54	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.85	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.63: Classification Report for Bi-LSTM with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.83	0.54	0.65	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.82	102000
weighted avg	0.98	0.98	0.98	102000

The One-hot encoding results are shown in Table 8.65, Table 8.66 and Table 8.67. Even though the AutoEncoder model failed to predict all the anomalies, the number of anomalies predicted are 90% true anomalies among the total prediction. Where as One-hot encoding and One-hot encoding with PCA both predicted 53% and 49% which are all true anomalies.

In case of Tf-idf the PCA predicted only 53% of the true anomalies but it did not predict any false positives. Where as AutoEncoder predicted 49% as anomalies and 99% of the prediction is true anomalies resulting in very low false positives. The results are shown in Table 8.68, Table 8.69 and Table 8.70

Table 8.64: Classification Report for Bi-LSTM with CountVectorizer and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.61	0.62	0.62	3737
Not Anomaly	0.99	0.99	0.99	98263
Accuracy			0.97	102000
Macro avg	0.80	0.80	0.80	102000
weighted avg	0.97	0.97	0.97	102000

Table 8.65: Classification Report for Bi-LSTM with One-Hot encoding

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.53	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.66: Classification Report for Bi-LSTM with One-Hot encoding and PCA

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.49	0.65	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.74	0.82	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.67: Classification Report for Bi-LSTM with One-Hot encoding and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.90	0.58	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.94	0.79	0.85	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.68: Classification Report for Bi-LSTM with Tf-idf

	Precision	Recall	F1-Score	Support
Anomaly	0.92	0.56	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.95	0.78	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.69: Classification Report for Bi-LSTM with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.53	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.70: Classification Report for Bi-LSTM with Tf-idf and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.99	0.49	0.65	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.98	0.74	0.82	102000
weighted avg	0.98	0.98	0.98	102000

The Bi-LSTM with word2vec was able to capture only 53% of anomalies but it did not predict any false positives but the model failed to perform well on the PCA data. It also fails to capture all the anomalies.

Table 8.71: Classification Report for Bi-LSTM with Word2Vec

	Precision	Recall	F1-Score	Support
Anomaly	1.00	0.53	0.70	3737
Not Anomaly	0.98	1.00	0.99	98263
Accuracy			0.98	102000
Macro avg	0.99	0.77	0.84	102000
weighted avg	0.98	0.98	0.98	102000

8.2.6 Ensemble Learning

By examining each results of the individual models it is shown that IForest, COPOD, ECOD, LSTM performed well in predicting the anomalies using CountVectorizer and PCA. The second set of models that performed well in capturing anomalies and giving less number of false positive rate are IForest, COPOD, ECOD using Tf-idf and PCA. The results of combining these models for better prediction is shown in the Table 8.74 and Table 8.75.

Table 8.72: Classification Report for Bi-LSTM with Word2Vec and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.75	0.62	0.68	3737
Not Anomaly	0.99	0.99	0.99	98263
Accuracy			0.98	102000
Macro avg	0.87	0.80	0.83	102000
weighted avg	0.98	0.98	0.98	102000

Table 8.73: Classification Report for Bi-LSTM with Word2Vec and AutoEncoder

	Precision	Recall	F1-Score	Support
Anomaly	0.59	0.91	0.71	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.97	102000
Macro avg	0.79	0.94	0.85	102000
weighted avg	0.98	0.97	0.98	102000

Table 8.74: Classification Report for Ensemble with CountVectorizer and PCA

	Precision	Recall	F1-Score	Support
Anomaly	0.82	1.00	0.89	3737
Not Anomaly	1.00	0.98	0.99	98263
Accuracy			0.99	102000
Macro avg	0.91	0.99	0.94	102000
weighted avg	0.99	0.99	0.99	102000

Table 8.75: Classification Report for Ensemble with Tf-idf and PCA

	Precision	Recall	F1-Score	Support
Anomaly	**0.84**	**1.00**	**0.91**	3737
Not Anomaly	1.00	0.99	1.00	98263
Accuracy			0.99	102000
Macro avg	0.92	1.00	0.95	102000
weighted avg	0.99	0.99	0.99	102000

8.3 Evaluation

This section explains about the evaluation of the model used in the thesis. The evaluation metrics used for the considered models are Precision, Recall, F1-score, time taken for training, the resources used and the expert interview has been done to check the predicted data to be true anomaly. Based on the experts review and by adding the additionally collected anomalous data from real-time cloud logs, the evaluation metrics were calculated on the unseen data. The accuracy can not be taken as a primary evaluation metric due to the class imbalance problem but the accuracy is also considered. The run-time performance is also been evaluated. The data collected are huge in number(millions). Even though the training set used is the unlabelled real-time data, the assumption is that the percentage of the anomalous data is very small when compared to the normal data. The time and resources are considered here because SysEleven does not provide GPU. Hence each model must be run using CPU. The training and predicting time required is considered due to the daily incoming of huge number of logs and all of them must go through the model to continuously monitor the cloud health.

The models uses different feature engineering methods and all the models are evaluated individually and also used the ensemble learning to choose the best fit. After experimenting with each case the model results are shown and explained Section 8.2. The best fit is determined based on the models ability to capture all the anomalies in the dataset because, false positives are affordable in this business problem but resulting in the huge number of false negative could lead to the issues such as production down, not able to meet the SLA and so on.

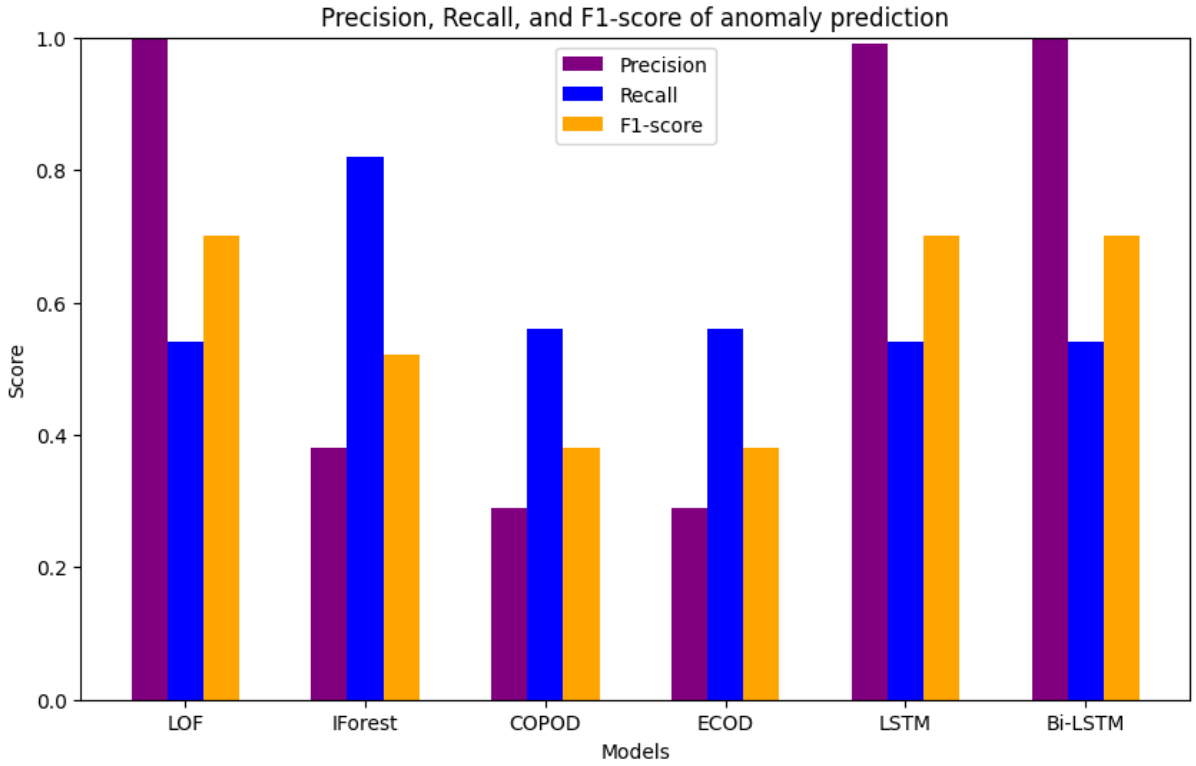


Figure 8.1: Precision, Recall, F1-score of anomaly prediction of each individual model using CV

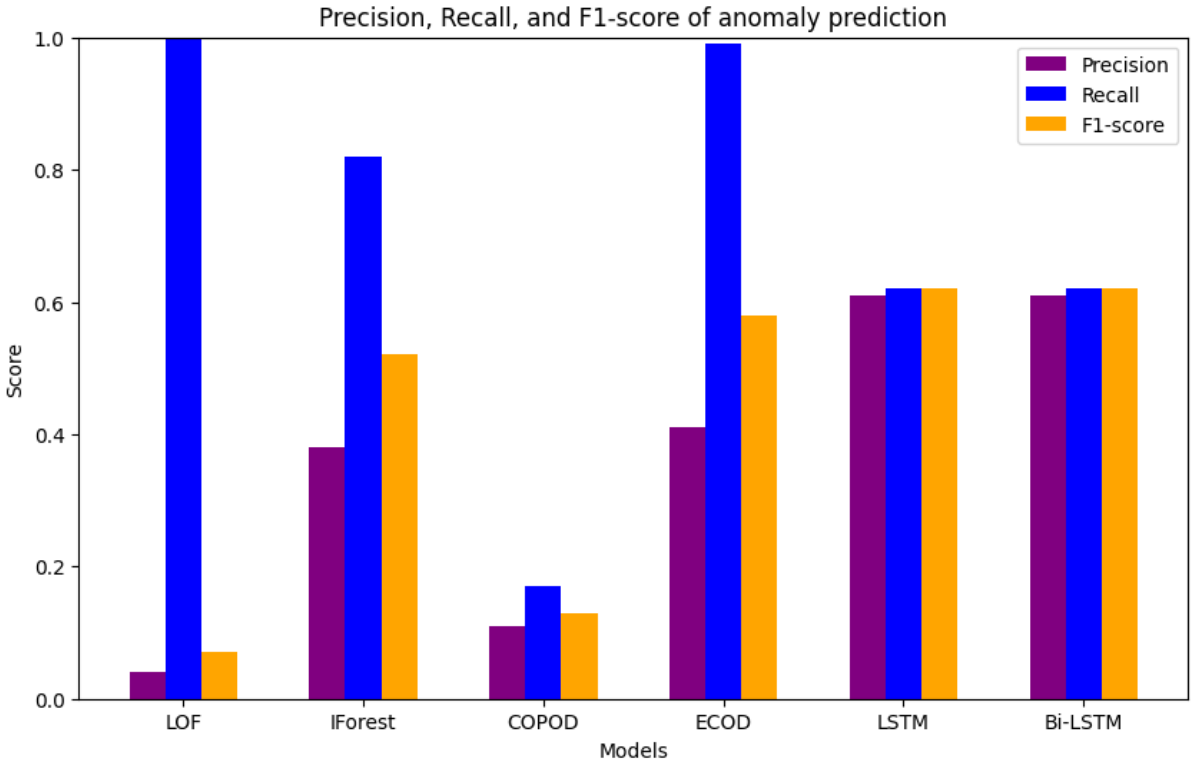


Figure 8.2: Precision, Recall, F1-score of anomaly prediction of each individual model using CV and AutoEncoder

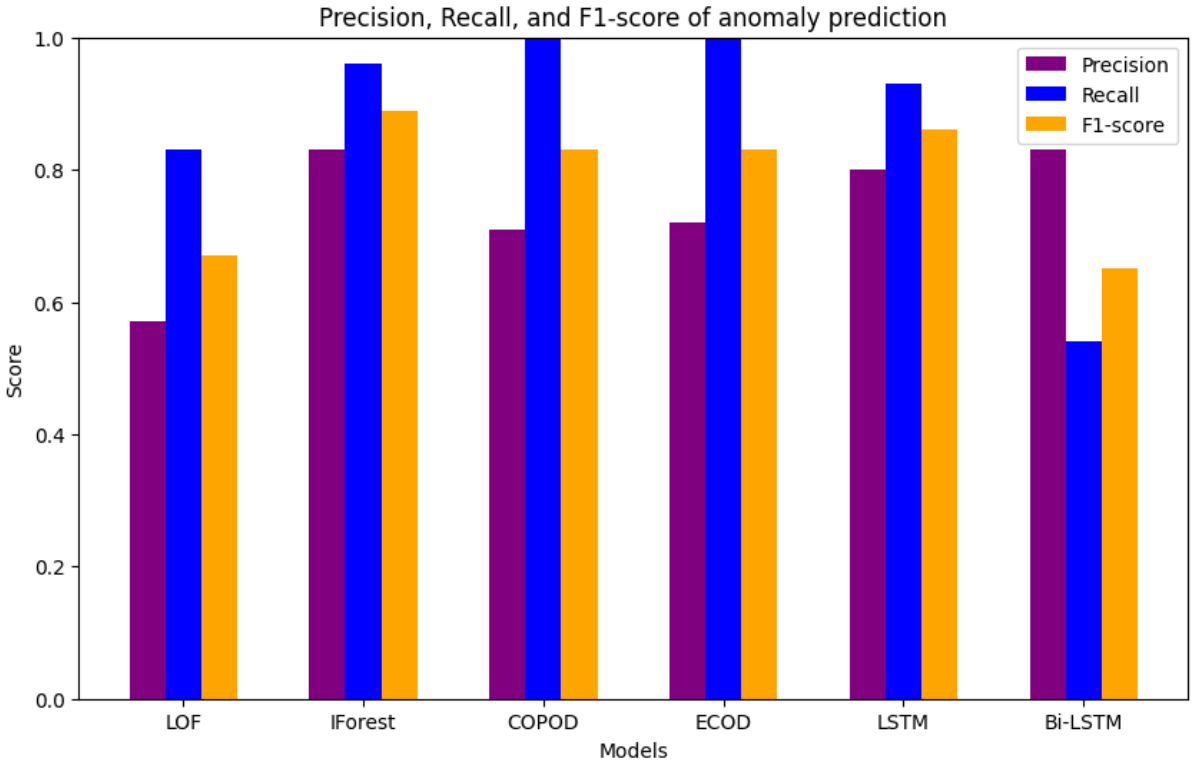


Figure 8.3: Precision, Recall, F1-score of anomaly prediction of each individual model using CV and PCA

The Figure 8.3 shows the evaluation metrics of each model individual prediction of anomalies. It can be seen that all the models performed better when compared to the model using all the features extracted by Countvectorizer and Countvectorizer with AutoEncoder as shown in the Figure 8.1 and Figure 8.2. As seen in the Figure 8.2 the LOF for AutoEncoder was able to predict all the anomalies, but the false positives in the prediction is very high. After the PCA on the Countvectorized data, the model's performance has improved and the COPOD and ECOD were able to predict all the anomalies and the false positive rate also decreased. The evaluation set includes the manual entries of anomalies by collecting them from the logs and by expert interview. The total samples used are 102000 and the number of anomalies manually included in this set is about 3.66%. The anomalies included in the evaluation set consists different cases. The COPOD and ECOD were able to capture all of them. Even though the false positive rate for these model were higher in number when compared to IForest and LSTM models after the PCA, they did not predict any false negative. But IForest and LSTM predicted some anomalies as not-anomaly.

Another case that resulted in having good prediction is Tf-idf and PCA. The performance of each model before and after the PCA and just by using Tf-idf can be seen in Figure 8.4, Figure 8.6, Figure 8.5. The Tf-idf and PCA feature engineering techniques were able to capture 100% of the anomalies present in the model but the false positive rate is still there and is a bit higher when compared to the CountVectorizer with PCA. LOF, IForest, COPOD and ECOD were able to capture all the anomalies after performing Tf-idf and PCA. The LOF did capture all the anomalies but the false positive rate is very high in this case.

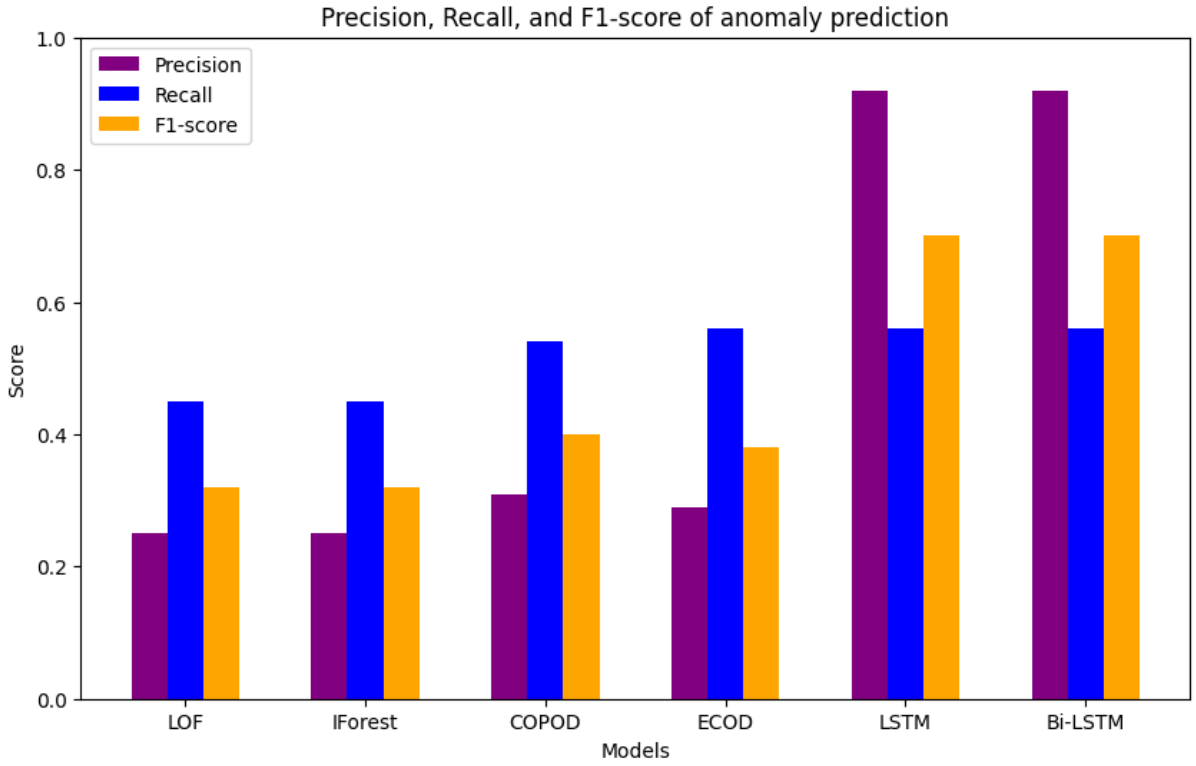


Figure 8.4: Precision, Recall, F1-score of anomaly prediction of each individual model using Tf-idf

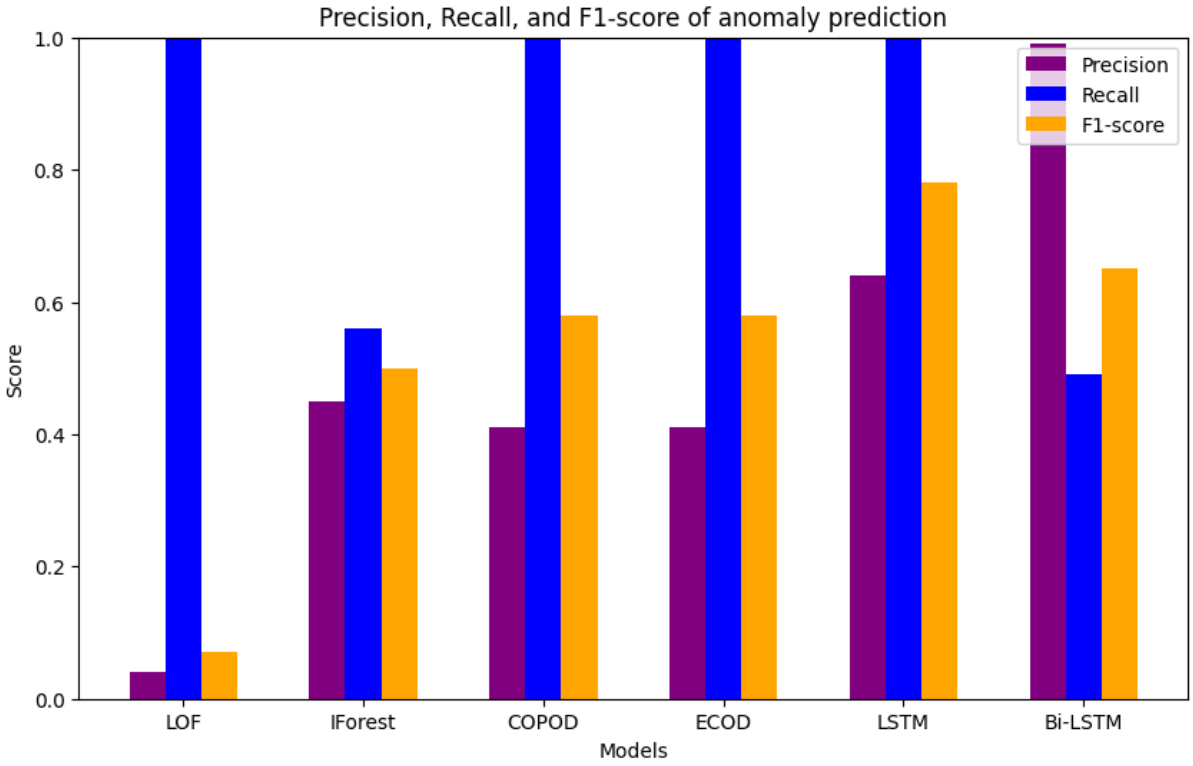


Figure 8.5: Precision, Recall, F1-score of anomaly prediction of each individual model using Tf-idf and AutoEncoder

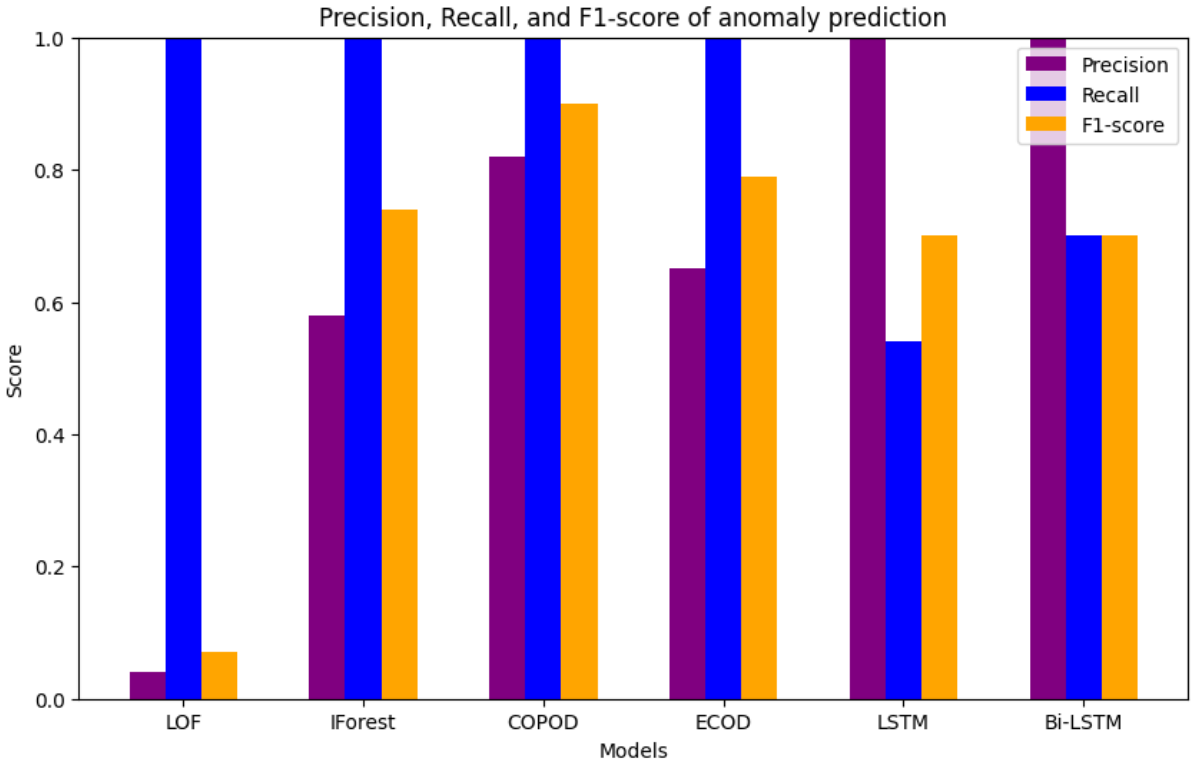


Figure 8.6: Precision, Recall, F1-score of anomaly prediction of each individual model using Tf-idf and PCA

The ensemble model was trained on the Tf-idf and CountVectorized data along PCA for the dimensionality reduction because the models such as IForest, COPOD, ECOD, LSTM performed well on these cases where they captured all of the anomalies. The Ensemble1 has IForest, COPOD, ECOD, LSTM and Ensemble2 has IForest, COPOD, ECOD. The performance of Ensemble model is show in the Figure 8.7. The models were experimented on other cases but there was not much improvement in the prediction and some even reduced the recall value than the individual prediction. The time consumed for training Ensemble model was approximately 13 minutes for training 1 million rows and 2 minutes for predicting 100k rows for Ensemble1 and it took about 3 minutes approximately to train and predict using Ensemble2. The Ensemble model follows the voting procedure to predict the anomalies. It will take the maximum votes for each rows and predicts the final prediction. After performing the ensemble learning it is observed that in Ensemble1 there was not much change when compared to individual prediction with COPOD. But in case of Ensemble2 the false positive rate was reduced because the model that might have predicted a false positive have not been predicted as false positive by other two models.

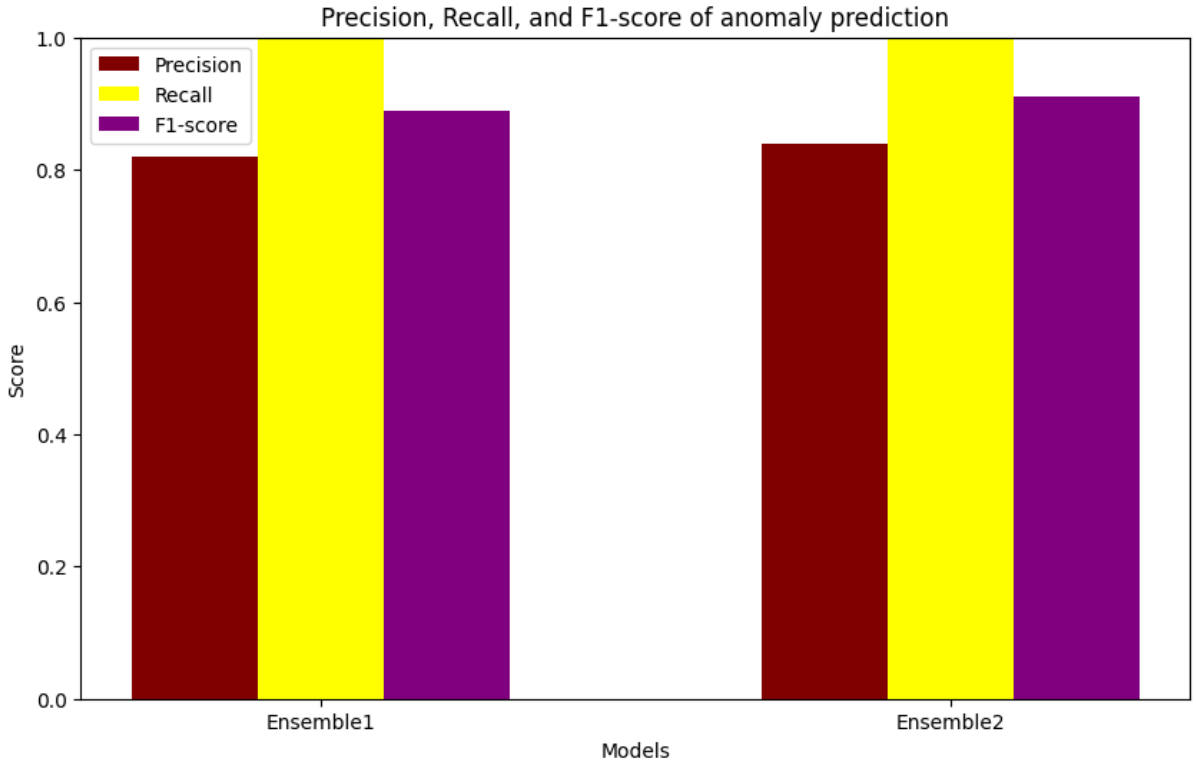


Figure 8.7: Precision, Recall, F1-score of anomaly prediction of each individual model using Ensemble1 and Ensemble2

The Figure 8.8 shows the approximate time taken by the models to train the dataset of 1 million rows. This was varied in different components of OpenStack because of the varied number of samples in the dataset. Out of all the models LOF took approximately 1 hour to train the models after the feature reduction. Before feature reduction LOF took on an average 2 hours for training and 40 minutes for predicting the 100k samples. The other models took very less time when compared to the LOF. The calculation of results were done for 1 million rows because the number of logs usually generated for each OpenStack API is approximately 1 million.

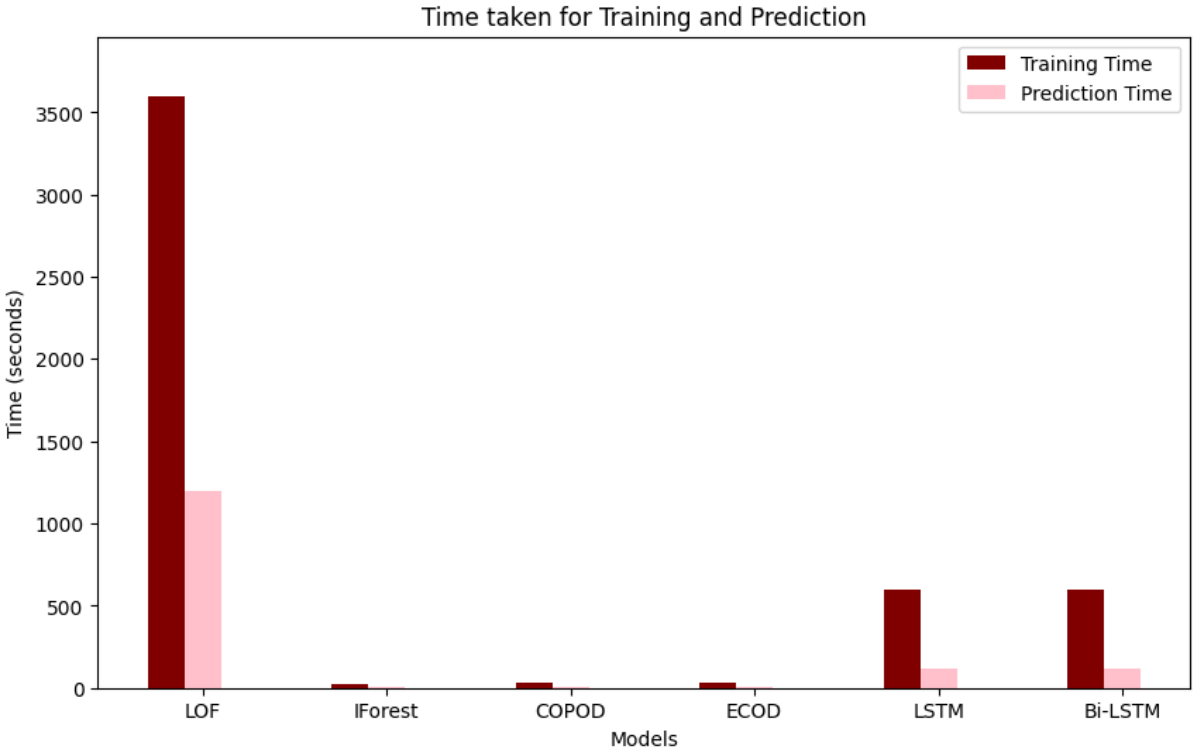


Figure 8.8: Training and Prediction time of models

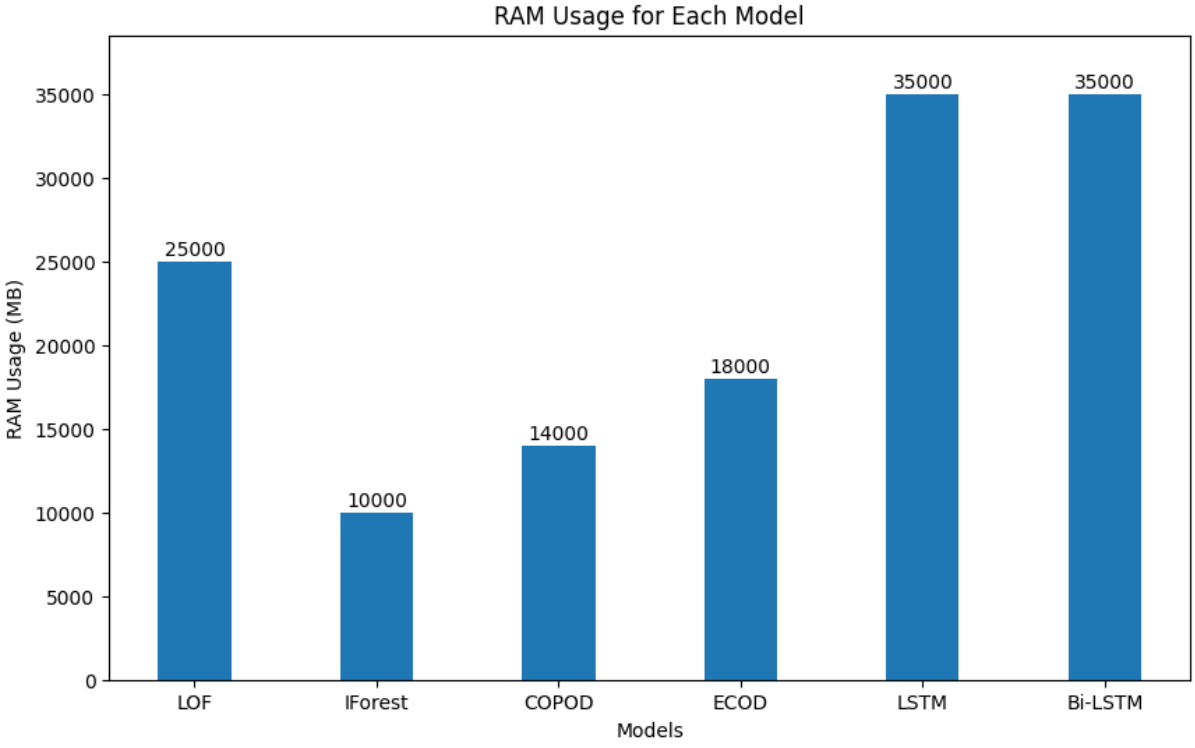


Figure 8.9: RAM Usage of Models

The Figure 8.9 shows the RAM usage by each model when training and predicting the 1 million rows. The LSTM and Bi-LSTM needed more compute resource when compared to the other model.

SREs from Cloustackers team were interviewed to verify the results of the prediction and based on their review, an unseen data was created for model evaluation which has been used to evaluate the models.

After the evaluation, it is clear that the LOF does not perform well for detecting anomalies, because of the amount of false positives it generated, its failure to capture all the anomalies and also the amount of time it takes for training. Different feature extraction techniques were tested in the thesis including the Word2Vec, but the techniques that were suited in this case are CountVectorizer and Tf-idf. Later when the ensemble method was used it is shown that the Tf-idf along with PCA was proved to be best fit for the data selected and the problem definition as it was able to capture all the anomalies and reduced the number of false positives when compared to other models.

9 Conclusion and Future Work

In this, thesis different Machine Learning models that were state of the art and have been used in detection of anomalies in OpenStack were discussed, compared and evaluated for the monitoring of logs to detect the anomalies. The data predicted were verified by experts and with the help of the ground truths collected the models were evaluated. The details of the evaluation results to the defined research questions are described in chapter along with the limitations and the potential future work are discussed.

9.1 Conclusion

This section will explain about the answer for the research questions defined in the thesis. The answers to the subquestions will also be discussed here in order to precisely answer the primary research question.

[RQ1:] Are the chosen machine learning algorithm are able to accomplish the business use case?

[SRQ1.1:] Which state of the art or novel algorithm is considered as the best fit in comparison to the other algorithm of choice? In order to find the answer for this question, many approached were experimented in the thesis. In order to train these models, the textual data needed to be represented in a way so that the ML model can process it for training as the machine learning models do not understand plain language. Hence the textual data were represented as vectors using the feature engineering techniques. The best suited representation are CountVectorizer and Tf-idf Vectorizer. But using simply these methods did not yield a good performance. Hence the feature dimension was reduced using PCA and Autoencoder, among which PCA was well suited and it captured all the information among the components. This allowed efficient training of models. After the training and the experiments, the novel algorithms COPOD and ECOD performed better than Iforest, LSTM and LOF for the dataset selected.

[SRQ1.2:] How the models are evaluated and what are the qualitative and quantitative assessment being considered? The models are evaluated using the evaluation metrics and the expert review. The SREs from SysEleven were interviewed to verify the results. Based on their review and by collecting the additional known anomaly logs from the OpenStack services, a dataset was created as a evaluation dataset which was used to get the Precision, Recall, F1-score, time and compute resources. The accuracy is not considered because of the class imbalance problem in the real world dataset. Using these metrics the model was evaluated. The importance is given to the model that was able to capture all the anomalies in the evaluation dataset. Because in case of cloud service providers, the service is being rendered to many businesses who host their applications

on the cloud. When anomalies go undetected or gets wrongly detected would make the production to go down and also will affect the SLA. In turn would cost the business. Hence false positives can be afforded but not false negatives. The time is also a important evaluation factor due to the number of logs generated. Every log must go through the model and if there are delays these logs would get stacked up and would cause memory issues as well as delay in detecting the anomalies. The compute resource is considered to evaluate the cost effectiveness of the model. In this case it is shown that for about 1 million rows average RAM size needed is approximately 20GiB.

Answer for [RQ1]: Based on the two answers in SRQ1.1 and SRQ1.2, the models captured all the anomalies in one of the cases at least. Even though LOF performed poorly in comparison with other models, it captured all the anomalies in case of Tf-idf with Autoencoder. The models Iforest, COPOD, ECOD performed well and LSTM and Bi-LSTM were performing poorly in some cases in terms of false negative and false positive.

[RQ2:]How the ensemble approach behaves on the detection of anomalies and which algorithms contribute towards the better performance?

[SRQ2.1:] Which is the best ensemble model that would fit the solution? After training the individual models, the IForest, COPOD, ECOD, LSTM performed well in capturing all the anomalies in the dataset with less false positives when compared to other models in case of CountVectorizer and PCA. The models IForest, COPOD, ECOD performed well individually in case of Tf-idf and PCA for the dataset with less false positives when compare to other models. Hence two ensemble models were selected for further tarining. Ensemble1 had a combination of IForest, COPOD, ECOD, LSTM combination. Whereas Ensemble2 had a combination of IForest, COPOD, ECOD. After training them with CountVectorizer and PCA and Tf-idf and PCA respectively, the Ensemble2 performed better when compared to Ensemble1 where it predicted all the anomalies with less false positives compared to Ensemble1. For further verification other combinations were tried, but their performance did not improve as some failed to capture all the anomalies and some had more false positives and also failed to predict all the anomalies. Hence the Ensemble2 is selected as the best performing model for the OpenStack dataset selected.

[SRQ2.2:] How can the ensemble models be evaluated? The models selected are evaluated by taking the same evaluation dataset used for the individual models evaluation which consisted of the anomalies included by collecting them in logs and based on expert interview. The models final prediction is decided based on the counting of maximum prediction received for each sample. The maximum prediction label received for the sample is selected as a final prediction for the model and it is evaluated by comparing it to the ground truth and also by consulting the experts. The importance and weightage is given to model that was able to capture all the anomalies and have less false positives. Hence Ensemble2 is selected as the best performing models as its recall is 100% and its precision is 84% which is higher when compared to other models. The models precision is 84% because the maximum prediction is the Ensemble2 is taken and this reduced the false positives as the 2 models prediction suppressed the other models prediction.

Answer for [RQ2]: Based on the experiments conducted and the answers provided in the sub-questions SRQ2.1 and SRQ2.2, it is clear that the Ensemble2 model that has the combination of IForest, COPOD, ECOD performs better when Tf-idf and PCA is applied to the dataset when compared to the other models.

9.2 Summary

The study was conducted to address the use case of building an anomaly detection model for monitoring and alerting for OpenStack logs using Machine Learning. Several models were experimented in the thesis which involved state of the art, novel and most used for anomaly detection in OpenStack logs. The DSR is the methodology that has been followed in the thesis which has helped to present the solution in a systematic way. The summary of the study with respect to DSR is explained in Table 9.1

The Machine Learning has been used to automate the process of detecting anomalies in the OpenStack logs by performing the comparative study of the algorithms such as LOF, IForest, COPOD, ECOD, LSTM/Bi-LSTM and the Ensemble model. The primary focus of the study was to develop a solution to the detection of anomalies in OpenStack logs that would reduce the human efforts in monitoring the cloud health. This has been done in a systematic way using the DSR.

The models were tested against each other to determine the bestfit and the Ensemble2 (IForest, COPOD, ECOD) proved to be the best fit for the OpenStack data selected and its recall is 100% and precision is 84% indicating that the model was able to detect all the anomalies and it detected less number of false positives when compared to the other models. Several feature extraction techniques including Word2Vec were tested in the thesis but the satisfactory results were obtained using Tf-idf and PCA with the Ensemble2 model.

9.3 Limitations and Challenges

The major limitation is the dataset that is available for training. The available open-source dataset for OpenStack logs are very limited which contains approximately 2k logs. The dataset used is a real time dataset which is collected over a period to train the model. The amount of incidents, issues that were available in the real time data are very small. Hence some of the anomalous data are collected from expert interview and an evaluation dataset has been formed to evaluate the models performance.

The second limitation is the amount of false positives that were detected by the model. As this depends on the data used for training and the less occurring logs are named as anomalous where in real time they are not. To determine such less occurring logs to apply data augmentation is a tedious task due to the amount of logs being generated and also due to the amount of logs being used for training.

The third limitation is determining the percentage of contamination in the dataset because of the nature of algorithms such as LOF, IForest, COPOD, ECOD. One should have the domain knowledge in order to accurately determine this and if dataset is very

Table 9.1: Summary of DSR phases in the thesis

Phases	**Description**
Environment	In this case the environment involves SysEleven Stack based on OpenStack which is used by 1000 plus customers across European Union, SREs who are involved in developing the cloud and maintaining it.
Business Objective	To build an automated anomaly detection system to eliminate the human efforts to detect the anomalies in order to suppress the method of manually defining the rules.
Research Focus	To demonstrate a Comparative Study of different machine learning algorithms such as LOF, IForest, COPOD, ECOD, LSTM/Bi-LSTM and Ensemble model in order to determin the best fit for detection anomalies in OpenStack logs.
Knowledge Base	A systematic research has been conducted to find the existing solutions in the environment and have created a model that could use ML for detecting the anomalies. The algorithms such as COPOD and ECOD are not yet experimented for the detection of anomalies to the best of our knowledge.
Developing	The solution has been developed by data collection, preprocessing, feature extraction, dimensionality reduction, model building, training, predicting the anomalies. The models trained are LOF, IForest, COPOD, ECOD, LSTM/Bi-LSTM and also Ensemble mdoel.
Evaluation	The models are evaluated using both qualitative and quantitative methods. The evaluation metrics such as Recall, Precision and F1-score are used for evaluation of models. The evaluation also included the expert interview to verify the detected anomalies. The models performance is compared against each other and the best fit is selected based on the models ability to detect the anomalies with less number of False positives.
Deployment	The final stage of the solution is to deploy the best fit ML model to cloud infrastructure so that it could monitor the incoming logs and notify the SREs by predicting the anomaly which has been done by building a pipeline using Fluentd between nodes and the server in which the model has been deployed.

large in size then it is a very difficult task even for the domain experts. Hence a trial and error method has been used in the thesis to determine the value of contamination.

Quality of the input data plays a major role in determining the anomalies and also to reduce the false positives being generated. Due to the class imbalance problem the performance of models in several cases were very low. Hence with the quality input data the models performance can be improved even better.

9.4 Future Work

The thesis focused on providing a systematic solution for detecting anomalies in the OpenStack logs using Machine Learning. The selected algorithms for predicting anomalies did perform better for the dataset used, but there is still room for improvement in reducing the false positives. Hence efforts must be made to accumulate the data that results in false positive and such data can be used in the training by performing data augmentation.

There are several deep learning algorithms available which can be used for detecting the anomalies and they can be explored given there are powerful resources available to train the models and use them for prediction.

Since the generation of logs are happening in a continuous fashion, online learning can be experimented in the future to have the model utilize the incoming logs and train.

The deployment does not have an user interface developed to monitor these predictions and also the generated predictions can be segregated in to low, medium and high priority based on the threshold. Hence an interactive UI can be developed to make the monitoring of the cloud health more effective.

Bibliography

[AAA+20] Matthew Akanle, Sunday Ajala, Emmanuel Adetiba, Funmilayo Moninuola, Victor Akande, Joke Badejo, and Ezekiel Adebiyi. Experimentations with openstack system logs and support vector machine for an anomaly detection model in a private cloud infrastructure. 08 2020. doi:10.1109/icABCD49160.2020.9183878.

[AAFAQ21] Zain Ashi, Mohammad Al-Fawa'reh, Laila Aburashed, and Malik Qasaimeh. Fast and reliable ddos detection using dimensionality reduction and machine learning. 02 2021.

[adb] ADBench: Anomaly Detection Benchmark, author=Songqiao Han and Xiyang Hu and Hailiang Huang and Minqi Jiang and Yue Zhao, booktitle=Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, year=2022, url=https://openreview.net/forum?id=foA$_S$$FQ9zo$0.

[BKG20] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 03 2020.

[BKNS00] Markus Breunig, Peer Kröger, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000. doi:10.1145/342009.335388.

[CLG+21] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael Lyu. Experience report: Deep learning-based system log analysis for anomaly detection. 07 2021.

[CLL22] Yiyong Chen, Nurbol Luktarhan, and Dan Lv. Logls: Research on system log anomaly detection method based on dual lstm. *Symmetry*, 14:454, 02 2022. doi:10.3390/sym14030454.

[CZL+20] Rui Chen, Shenglin Zhang, Dongwen Li, Yuzhe Zhang, Fangrui Guo, Weibin Meng, Dan Pei, Yuzhi Zhang, Xu Chen, and Yuqing Liu. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 37–47, 2020. doi:10.1109/ISSRE5003.2020.00013.

[DSM19] Gleen A. Dalaorao, Ariel M. Sison, and Ruji P. Medina. Integrating collocation as tf-idf enhancement to improve classification accuracy. In *2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pages 282–285, 2019. doi:10.1109/TSSA48701.2019.8985458.

[FFZ21] Markus Fält, Stefan Forsström, and Tingting Zhang. Machine learning based anomaly detection of log files using ensemble learning and self-attention. In *2021 5th International Conference on System Reliability and Safety (ICSRS)*, pages 209–215, 2021. doi:10.1109/ICSRS53853.2021.9660694.

[FG19] Amir Farzad and T. Aaron Gulliver. Log message anomaly detection and classification using auto-b/lstm and auto-gru, 11 2019.

[FG20a] Amir Farzad and T. Aaron Gulliver. Log message anomaly detection with oversampling. *International Journal of Artificial Intelligence Applications*, 11:53–65, 07 2020. doi:10.5121/ijaia.2020.11405.

[FG20b] Amir Farzad and T. Aaron Gulliver. Unsupervised log message anomaly detection. *ICT Express*, 6:229–237, 09 2020. doi:10.1016/j.icte.2020.06.003.

[Flu] Fluentd contributors. Fluentd. [Online; accessed 9-March-2023]. URL: https://www.fluentd.org/.

[Fra] François Torregrossa, Robin Allesiardo, Vincent Claveau, Nihel Kooli Guillaume Gravier . A survey on training and evaluation of word embeddings. URL: https://link.springer.com/article/10.1007/s41060-021-00242-8#Sec2.

[GBK+22] Fabian Gerz, Tolga Bastürk, Julian Kirchhoff, Joachim Denker, Loui Al-Shrouf, and Mohieddine Jelali. A comparative study and a new industrial platform for decentralized anomaly detection using machine learning algorithms. 07 2022. doi:10.1109/IJCNN55064.2022.9892939.

[GKK+19] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y. Zomaya, and Rajiv Ranjan. A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3):924–935, 2019. doi:10.1109/TNSM.2019.2927886.

[Gra] Grafana contributors. Grafana alerting. [Online; accessed 9-March-2023]. URL: https://grafana.com/docs/grafana/latest/alerting/?plcmt=footer#alerting.

[Hev07] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[HHTM19] Ying-Feng Hsu, ZhenYu He, Yuya Tarutani, and Morito Matsuoka. Toward an online network intrusion detection system based on ensemble learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 174–178, 2019. doi:10.1109/CLOUD.2019.00037.

[HK21] Tanja Hagemann and Katerina Katsarou. A systematic review on anomaly detection for cloud computing environments. In *2020 3rd Artificial Intelligence and Cloud Computing Conference*, AICCC 2020, page 83–96, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3442536.3442550.

[HLF+20] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. Hitanomaly: Hierarchical transformers for anomaly detection in system log. *IEEE Transactions on Network and Service Management*, PP:1–1, 10 2020. doi:10.1109/TNSM.2020.3034647.

[HO21] Mohammad Hamayel and Amani Owda. A novel cryptocurrency price prediction model using gru, lstm and bi-lstm machine learning algorithms. *AI*, 2:477–496, 10 2021. doi:10.3390/ai2040030.

[HWZ+21] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. Log-based anomaly detection with robust feature extraction and online learning. *IEEE Transactions on Information Forensics and Security*, 16:2300–2311, 2021. doi:10.1109/TIFS.2021.3053371.

[HZZL17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40, 2017. doi:10.1109/ICWS.2017.13.

[IPZ+21] Mohammad Saiful Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. Anomaly detection in a large-scale cloud platform. pages 150–159, 05 2021. doi:10.1109/ICSE-SEIP52600.2021.00024.

[LLC21] Dan Lv, Nurbol Luktarhan, and Yiyong Chen. Conanomaly: Content-based anomaly detection for system logs. *Sensors*, 21:6125, 09 2021. doi:10.3390/s21186125.

[LNZ+21] Xinqiang Li, Weina Niu, Xiaosong Zhang, Runzi Zhang, Zhenqi Yu, and Zimu Li. Improving performance of log anomaly detection with semantic and time features based on bilstm-attention. In *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*, pages 661–666, 2021. doi:10.1109/CECIT53797.2021.00121.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi:10.1109/ICDM.2008.17.

[LZ22] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. 01 2022. doi:10.1109/ASE51524.2021.9678773.

[LZB+20] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: Copula-based outlier detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123, 2020. doi:10.1109/ICDM50108.2020.00135.

[Man22] Soumen Manna. Small sample estimation of classification metrics. In *2022 Interdisciplinary Research in Technology and Management (IRTM)*, pages 1–3, 2022. doi:10.1109/IRTM54583.2022.9791645.

[MST+17] Sidharth Mishra, Uttam Sarkar, Subhash Taraphder, Sanjoy Datta, Devi Swain, Reshma Saikhom, Sasmita Panda, and Menalsh Laishram. Principal component analysis. *International Journal of Livestock Research*, page 1, 01 2017. doi:10.5455/ijlr.20170415115235.

[MZHM11] Adetokunbo Makanju, A. Zincir-Heywood, and Evangelos Milios. A lightweight algorithm for message type extraction in system application logs. *Knowledge and Data Engineering, IEEE Transactions on*, 24:1 – 1, 01 2011. doi:10.1109/TKDE.2011.138.

[NCK19a] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. Anomaly detection and classification using distributed tracing and deep learning. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 241–250, 2019. doi:10.1109/CCGRID.2019.00038.

[NCK19b] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. Anomaly detection from system tracing data using multimodal deep learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 179–186, 2019. doi:10.1109/CLOUD.2019.00038.

[Opea] OpenStack contibutors. Openstack architecture. [Online; accessed 9-March-2023]. URL: https://docs.openstack.org/nova/pike/install/overview.html.

[Opeb] OpenStack contributors. Openstack. [Online; accessed 9-March-2023]. URL: https://www.openstack.org/software/.

[Opec] OpenStack contributors. Openstack by redhat. [Online; accessed 9-March-2023]. URL: https://www.redhat.com/en/topics/openstack.

[Oped] OpenStack Contributors. Openstack networking. [Online; accessed 9-March-2023]. URL: https://docs.openstack.org/security-guide/networking/architecture.html.

[Pro] Prometheus contributors. Prometheus. [Online; accessed 9-March-2023]. URL: https://prometheus.io/docs/introduction/overview/.

[RFA21] Yousra Regaya, Fodil Fadli, and Abbes Amira. Point-denoise: Unsupervised outlier detection for 3d point clouds enhancement. volume 80, pages 1–17, 07 2021. doi:10.1007/s11042-021-10924-x.

[RWJ22] Piotr Ryciak, Katarzyna Wasielewska, and Artur Janicki. Anomaly detection in log files using selected natural language processing methods. *Applied Sciences*, 12:5089, 05 2022. doi:10.3390/app12105089.

[SMJ+22] Faisal Shahzad, Abdul Mannan, Abdul Rehman Javed, Ahmad Almadhor, Thar Baker, and Dhiya Al-Jumeily Obe. Cloud-based multiclass anomaly detection and categorization using ensemble learning. *Journal of Cloud Computing*, 11:74, 11 2022. doi:10.1186/s13677-022-00329-y.

[SvB12] Christian Sonnenberg and Jan vom Brocke. Evaluations in the science of the artificial – reconsidering the build-evaluate pattern in design science research. In Ken Peffers, Marcus Rothenberger, and Bill Kuechler, editors, *Design Science Research in Information Systems. Advances in Theory and*

Practice, pages 381–397, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Sys] SysEleven contributors. Syseleven stack. [Online; accessed 9-March-2023]. URL: https://www.syseleven.de/produkte-services/openstack-cloud/.

[SZLNV+22] Carmen Sánchez-Zas, Xavier Larriva-Novo, Víctor A. Villagrá, Mario Sanz Rodrigo, and José Ignacio Moreno. Design and evaluation of unsupervised machine learning models for anomaly detection in streaming cybersecurity logs. *Mathematics*, 10(21), 2022. URL: https://www.mdpi.com/2227-7390/10/21/4043.

[Ver21] Arthur Vervaet. Monilog: An automated log-based anomaly detection system for cloud computing infrastructures. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2739–2743, 2021. doi:10.1109/ICDE51399.2021.00317.

[YAS+19] Yue Yuan, Han Anu, Wenchang Shi, Bin Liang, and Bo Qin. Learning-based anomaly cause tracing with synthetic analysis of logs from multiple cloud service components. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 66–71, 2019. doi:10.1109/COMPSAC.2019.00019.

[YM20] Tharindu Yasarathna and Lankeshwara Munasinghe. Anomaly detection in cloud network data. 09 2020. doi:10.1109/SCSE49731.2020.9313014.

[YP21] Geeta Yadav and Kolin Paul. Global monitor using spatiotemporally correlated local monitors. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, pages 1–10, 2021. doi:10.1109/NCA53618.2021.9685330.

[YSLQ19] Yue Yuan, Wenchang Shi, Bin Liang, and Bo Qin. An approach to cloud execution failure diagnosis based on exception logs in openstack. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 124–131, 2019. doi:10.1109/CLOUD.2019.00031.

[ZQZ+22] Junwei Zhou, Yijia Qian, Qingtian Zou, Peng Liu, and Jianwen Xiang. Deepsyslog: Deep anomaly detection on syslog using sentence embedding and metadata. *IEEE Transactions on Information Forensics and Security*, PP:1–1, 01 2022. doi:10.1109/TIFS.2022.3201379.

[ZWL+20] Pengpeng Zhou, Yang Wang, Zhenyu Li, Xin Wang, Gareth Tyson, and Gaogang Xie. Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2020. doi:10.1109/IWQoS49365.2020.9212954.

[ZXH+22] Zhuping Zou, Yulai Xie, Kai Huang, Gongming Xu, Dan Feng, and Darrell Long. A docker container anomaly monitoring system based on optimized isolation forest. *IEEE Transactions on Cloud Computing*, 10(1):134–145, 2022. doi:10.1109/TCC.2019.2935724.