Assignment of master’s thesis

Title: Log Server Analytics
Student: Bc. Daniil Fedotov
Supervisor: doc. Ing. Tomáš Vitvar, Ph.D.
Study program: Informatics
Branch / specialization: Web and Software Engineering, specialization Web Engineering
Department: Department of Software Engineering
Validity: until the end of summer semester 2021/2022

Instructions

The goal of the master thesis is to develop methods for analysis of server logs to understand information provided by large logs generated by servers during runtime. Follow the below guidelines.

1. Analyze the current state of art in log analytics and understand various approaches that are currently available on the market.
2. Describe underlying methods or algorithms you will use to develop log analytics.
3. Describe architecture of an environment with servers where log analytics will be performed.
4. Develop and implement at least two methods for log analytics and describe related log analytics processes.
5. Perform tests on large server logs and evaluate results generated by your methods.

______________________________

Master’s thesis

Log Server Analytics

Bc. Daniil Fedotov

Department of Software Engineering
Supervisor: doc. Ing. Tomáš Vitvar, Ph.D.

May 5, 2021
Acknowledgements

I would like to thank my supervisor doc. Ing. Tomáš Vitvar, Ph.D. for encouragement, patient guidance and advices he has provided throughout the work.
I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 5, 2021
Abstrakt

V současné době informační systémy z pohledu infrasktury zahrnují operační systémy, web servery, aplikační servery, prvky pro vyvažování zátěže a sítovou komunikaci, které produkují velké množství provozních dat a zapisují je do logů. Analýza takových logů pomáhá pochopit současný stav systému – například, určit místa, kde vznikají chyby. Podobné znalostí pomáhá vylepšit výkonnost, stabilitu a dostupnost systému. Člověk není shopen přečíst, analyzovat a zpracovat velký počet zpráv v rozumném čase, tedy má smysl provést sdružovací analýzu nad logovými soubory ze serverů. Výsledkem takové analýzy budou skupiny (shluky) podobných logových zpráv. Tento přístup důvěryhodně snížit dimenzi původních dat logu a analyzovat skupiny místo jednotlivých zpráv logu, což zjednoduší vyhledávání a eliminaci problémů, které vznikaly během provozu serveru.

Tato diplomová práce návrhuje software pro sdružovací analýzu serverových logů, založený na metodách zpracování přirozeného jazyka, strojově-generovaných textech a algoritmech strojového učení s následující analýzou shluků logových dat.

Abstract

Currently, information systems from the infrastructural point of view containing operating systems, web servers, application servers, load balancers, network communication elements produce a huge amount of operations data and write it to logs. Analysis of such logs can help to understand the current state of the system – for instance, to determine the places, when errors occur. Such a knowledge helps to improve the performance of the system, make it more stable and accessible. Human is not able to read, analyze and process such a number of messages in a reasonable time, so it makes sense to perform a cluster analysis on server log files. Several groups (clusters) of similar log messages will be generated after clustering. This approach allows to significantly reduce the dimension of original log data, and allows to analyze not individual log entries, but groups, which simplifies searching and elimination of problems that have arisen during the server runtime.

This thesis proposes software for server log clustering based on natural language processing and machine-generated text processing techniques and machine learning algorithms, followed by the analysis of clustered log data.

Keywords Log Analytics, Log Processing, Log Data, Log Analytics Systems, Data Analytics, Data Science, Natural Language Processing, Machine Learning, Cluster Analysis.
## Contents

1. **Introduction** 1
   - Motivation .................................................. 1
   - Goal of the thesis ......................................... 1
   - Thesis structure ........................................... 2

2. **Basic concepts of Log Analysis** 3
   - Log Analysis ................................................ 3
   - Log Analysis core functions ............................... 3

3. **State-of-the-art** 5
   - Related works ............................................... 5
   - State-of-the-art systems .................................... 6
     - Splunk ...................................................... 6
     - ELK Stack .................................................. 7
     - Logalyze ..................................................... 8
     - Graylog ..................................................... 9
     - SolarWind Loggly ......................................... 9

4. **WebLogic log structure** 11

5. **Vectorization of Log Entries** 15
   - Natural Language Processing .............................. 15
     - Linguistic vectorization approach ....................... 16
     - Tokenization .............................................. 16
     - Dimensionality reduction and normalization ........... 17
     - Encoding ................................................... 18
       - Term Frequency Vectorization ......................... 18
       - One-Hot Vectorization .................................. 19
       - Term Frequency - Inverse Document Frequency ...... 20
     - Summary .................................................... 21
4.2 N-grams vectorization approach ............................ 22
  4.2.1 N-grams tokenization .................................. 22
  4.2.2 Dimensionality reduction and normalization .......... 22
  4.2.3 Encoding .................................................. 23
  4.2.4 Summary .................................................. 23

5 Cluster analysis ............................................... 25
  5.1 Cosine Similarity based clustering ...................... 25
  5.2 Machine Learning clustering ............................. 26
  5.3 Categories of machine learning methods ................. 27
    5.3.1 Supervised learning ................................ 27
    5.3.2 Unsupervised learning ............................... 27
    5.3.3 Semi-supervised learning ............................ 28
  5.4 Self-Organizing Map ...................................... 28
    5.4.1 Description .......................................... 28
    5.4.2 Euclidean distance .................................. 28
    5.4.3 Structure of the SOM ................................ 29
    5.4.4 Training algorithm .................................. 30
    5.4.5 Summary .............................................. 33
  5.5 Growing Self-Organizing Map ............................ 34
    5.5.1 Description .......................................... 34
    5.5.2 GSOM structure ...................................... 34
    5.5.3 Training algorithm .................................. 37
      5.5.3.1 Initialization phase ............................... 37
      5.5.3.2 Growing phase .................................... 37
      5.5.3.3 Smoothing phase .................................. 38
    5.5.4 GSOM clustering ...................................... 38
    5.5.5 Summary .............................................. 39

6 Implementation ............................................... 41
  6.1 Requirements ............................................. 41
    6.1.1 Non-functional requirements .......................... 41
    6.1.2 Functional requirements ................................ 42
  6.2 Use cases .................................................. 42
  6.3 Modules .................................................... 43
    6.3.1 LogEntry ............................................. 44
    6.3.2 Parser .................................................. 44
    6.3.3 Vectorization .......................................... 44
    6.3.4 Clustering ............................................. 44
    6.3.5 Neural .................................................. 45
    6.3.6 Reporting .............................................. 46
    6.3.7 Metrics ................................................ 47
    6.3.8 Evaluator .............................................. 48
    6.3.9 Main .................................................... 48
# 7 Evaluation and Experiments

## 7.1 Evaluation

### 7.1.1 Rand Index

## 7.2 Implemented clustering approaches

### 7.2.1 Cosine similarity based clustering with Linguistic vectorization

### 7.2.2 Cosine similarity based clustering with N-grams vectorization

### 7.2.3 GSOM clustering with Linguistic vectorization

### 7.2.4 GSOM clustering with N-grams vectorization

### 7.2.5 Evaluation summary

## 7.3 Correlation

### Conclusion

### Future works

### Bibliography

### A Acronyms

### B Contents of enclosed CD

### C Class diagram

### D Installation manual and user guide

#### D.1 Installation manual

##### D.1.1 Copy the application

##### D.1.2 Install required packages

#### D.2 Usage guide

##### D.2.1 Run the application

##### D.2.2 Cosine similarity based clustering

##### D.2.3 GSOM based clustering

#### D.3 Analysis of clusters
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>WebLogic server log entry pattern</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>WebLogic Server logging process</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>NLP Workflow</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>Term Frequency Vectorization</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>One-Hot Vectorization</td>
<td>19</td>
</tr>
<tr>
<td>4.4</td>
<td>Term Frequency - Inversed Document Frequency</td>
<td>20</td>
</tr>
<tr>
<td>4.5</td>
<td>Tokenization using uni-grams</td>
<td>22</td>
</tr>
<tr>
<td>4.6</td>
<td>Tokenization using bi-grams</td>
<td>23</td>
</tr>
<tr>
<td>4.7</td>
<td>Tokenization using tri-grams</td>
<td>23</td>
</tr>
<tr>
<td>4.8</td>
<td>Vectorization using bi-grams</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>Angle between two vectors $\mathbf{A}$ and $\mathbf{B}$</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>Euclidean distance between points $a$ and $b$</td>
<td>29</td>
</tr>
<tr>
<td>5.3</td>
<td>Structure of Self-Organizing Map with determined clusters</td>
<td>30</td>
</tr>
<tr>
<td>5.4</td>
<td>BMU’s neighborhood</td>
<td>32</td>
</tr>
<tr>
<td>5.5</td>
<td>The initial state of GSOM</td>
<td>34</td>
</tr>
<tr>
<td>5.6</td>
<td>New neurons growth</td>
<td>35</td>
</tr>
<tr>
<td>5.7</td>
<td>New neuron generation case 1</td>
<td>36</td>
</tr>
<tr>
<td>5.8</td>
<td>New neuron generation case 2</td>
<td>36</td>
</tr>
<tr>
<td>5.9</td>
<td>New neuron generation case 3</td>
<td>36</td>
</tr>
<tr>
<td>5.10</td>
<td>New neuron generation case 4</td>
<td>36</td>
</tr>
<tr>
<td>6.1</td>
<td>Use cases diagram</td>
<td>43</td>
</tr>
<tr>
<td>6.2</td>
<td>The workflow and modules of the developed log analytics application</td>
<td>43</td>
</tr>
<tr>
<td>7.1</td>
<td>Clustered evaluation dataset</td>
<td>51</td>
</tr>
<tr>
<td>7.2</td>
<td>Confusion matrix</td>
<td>53</td>
</tr>
<tr>
<td>7.3</td>
<td>Cosine similarity based clustering with linguistic vectorization</td>
<td>55</td>
</tr>
<tr>
<td>7.4</td>
<td>Cosine similarity based clustering with N-grams vectorization</td>
<td>56</td>
</tr>
<tr>
<td>7.5</td>
<td>GSOM with linguistic vectorization quantization error graph</td>
<td>57</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>7.6 Trained GSOM with linguistic vectorization</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>7.7 GSOM with linguistic vectorization clustering</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>7.8 GSOM with N-grams vectorization quantization error graph</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>7.9 Trained GSOM with linguistic vectorization</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>7.10 GSOM approach clustering</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>7.11 Server load graph</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>7.12 Log entries distribution among the clusters</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>7.13 Clusters containing over than 1000 log entries</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>7.14 Log entries time series constructed from observing clusters</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>7.15 Graphs comparison</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>7.16 Error clusters distribution in time</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>C.1 Class diagram of the developed application</td>
<td>82</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>WebLogic server log format description</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Message severity levels</td>
<td>13</td>
</tr>
<tr>
<td>7.1</td>
<td>Cosine similarity based clustering with linguistic vectorization</td>
<td>55</td>
</tr>
<tr>
<td>7.2</td>
<td>Cosine similarity based clustering with N-grams vectorization evaluation</td>
<td>56</td>
</tr>
<tr>
<td>7.3</td>
<td>GSOM clustering with linguistic vectorization evaluation</td>
<td>58</td>
</tr>
<tr>
<td>7.4</td>
<td>GSOM clustering with N-grams vectorization evaluation</td>
<td>59</td>
</tr>
<tr>
<td>7.5</td>
<td>Evaluation results of all considered methods</td>
<td>61</td>
</tr>
</tbody>
</table>
Introduction

Motivation

In our days each organization uses complex software systems that makes business easier and much more productive. Such software systems typically consist of a number of services, databases and application servers, which are connected via the network and operate with a huge amount of data. Supporting these systems is a very complicated task. Each part of the system helps administrators to monitor the system state through generating logs, which describe every single activity performing on each particular part of the entire system.

Obviously, logs provide us with very useful information that needs to be stored and analyzed to keep the system in working state. The software size and complexity is continuously increasing that leads to rapid growth of log data. These massive amounts of logs need to be effectively processed and analyzed. One of the most important tasks needs to be done with logs is clustering – partitioning log entries on several categories depending on the content of the message in the log entry.

Analysis of log data can be extremely helpful in situations, when many errors occur in the system.

Goal of the thesis

The goal of the master’s thesis is to develop a system that will perform clustering of logs generated by server. The found clusters will help to analyze and understand information provided by large logs generated by server. To reach the goal the following tasks need to be done:

- Define modern approaches in log server analytics, understand its concepts and principles,
- Explore the architecture of an environment with servers where log analytics will be performed,
**Introduction**

- Design and implement an application for log analytics that will be able to cluster log entries,
- Evaluate clustering approaches implemented in the application,
- Apply the developed application on real server log data to determine if there is a correlation between server load and the error clusters.

The tool should be provided as an installable package for the target systems.

**Thesis structure**

The thesis is into seven chapters. The first chapter describes the basic concepts of log analysis, defines the typical phases of log analysis process, and represents its core functions.

In the second chapter contains the description of related works and analysis of state-of-the-art log analytics systems including their advantages and disadvantages.

The third chapter describes the server infrastructure and structure of the logs produced by the server.

In the fourth chapter there are several vectorization techniques are presented. The main principles of natural language processing are described.

The fifth chapter describes the cluster analysis of log data. Cosine similarity based and machine learning approaches to log analytics are considered. The main types of machine learning are presented. Self-Organizing Map concept is defined along with its improved version, Growing Self-Organizing Map. The training algorithm is described.

The sixth chapter contains requirements for the application. Main application modules are described in detail.

In the seventh chapter implemented clustering techniques are applied on real server log data. Evaluation of all clustering approaches are performed. The clustering method is also applied for finding the correlation between server load and error clusters.
Chapter 1

Basic concepts of Log Analysis

1.1 Log Analysis

Log Analysis is a complex process of understanding of massive amounts of machine-generated texts, also known as log events or log entries, which provides useful information about how the components of the system works and what happens across the infrastructure. Typically, logs are streams of chronologically arranged messages which are generated by web servers, application servers, operation systems, elements of network communication etc. Log Analysis usually could be performed in the following several phases:

• **Collecting** – gathering all logs from all elements of the current infrastructure,

• **Centralization and indexing** – this phase deals with transitioning of collected data to a centralized logging platform, where log entries will be normalized to a common format to ensure uniformity. Also, the log entries can be indexed to make searching more efficient. The described process simplifies the further querying and analysis,

• **Searching and analysis** – searching for log entries which have a structure, defined by a pattern, or satisfied the queries,

• **Monitoring** – monitoring helps to detect the anomalies occurrences in system runtime and determine how them impacted the performance.

1.2 Log Analysis core functions

Log Analysis is a complex process that should follow the following functions:

• **Pattern Detection and Recognition** – filtering log entries based on a predefined patterns. Detecting patterns is an essential part of log analysis as it helps spot anomalies,
1. Basic concepts of Log Analysis

- **Log Normalization** – conversion of individual log entry elements such as timestamps, IP addresses, machine IDs, messages, etc. to a common format,

- **Classification and Tagging** – the process of tagging messages with keywords and categorizing them into separate classes. This allows to simplify and customize the further analysis and visualization,

- **Correlation Analysis** – finding correlations among logs from different sources of the system infrastructure. Analysis of such correlations helps in finding similar behavior of the elements of the system. This information is very helpful when an incident occurs somewhere in the system. For instance, in the case of malicious activity, it allows to filter and correlate logs coming from the network devices, firewalls, servers, and other sources. Correlation analysis is usually associated with alerting systems – based on the pattern you identified, you can create alerts for when the log analyzer discovers similar activity in logs,

- **Artificial Ignorance** – machine learning process that recognizes and discards log entries that are not useful. Typically, artificial ignorance is used to detect anomalies. That means to ignore routine messages generated from the normal operation of the system like regular system updates, thus labeling them as uninteresting. Artificial ignorance alerts about new and unusual events, even about common events that should have occurred but did not – for example, if a weekly updated has failed. Such an anomaly in system operation should be investigated [1, 2].
2.1 Related works

Log Analysis plays an important role in operations area. Automated processing of log message streams is practically relevant for the maintenance Service Oriented Architecture (SOA) environments.

Some of log processing techniques are based on clustering approaches. Tang et al. [3] proposed a method which finds the most representative message subsequences. The algorithm converts each log entry into a set of term pairs and then partition the log entries on converted term pairs, thus, uses these pairs as vectors. Such an approach can lead to highly dimensional vectors. Wurzenberger et al. [4] proposes incremental clustering method, sequential generation of clusters based on similarity measurement between log entries, with further semi-supervised anomaly detection. Similarity measurement is being performed on either string metrics or numerical distance metrics. Unlike the previous method in this work vectors’ dimensions are reduced with PCA technique, which can lead to losing the data. Using natural language processing methods, such as discarding stopwords, stemming and lemmatization has an advantage for dimensional reduction, since they preserve the semantic structure of the messages.

Cascading clustering approach is presented by He et al. [5], where the classical hierarchical agglomerative clustering is conducted on limited number of vectors obtained from log entries, then other unsampled input data is matched to obtained clusters.

Since log records have machine-generated nature Jiang et al. [6] proposed a Machine Language Processing approach, called P-gram, that deals with extraction of position-specific keywords in the log entries’ structure. Then these positional keywords can are used as features for the further K-Mediods log entries clustering.

Techniques used in Mapreduce frameworks might be helpful in log analytics area as well. The overview of Mapreduce versions of existing partition-based
clustering techniques such as K-means, K-prototypes, K-models, FCM is presented by Sardar et al. [7]. Teffer et al. [8] presented a Mapreduce method based on adaptive hash calculation for hierarchical clustering of streaming data. In this method the clustering via similarity metrics stands for map function, and the reduce operation defines the representative vector. These methods uses centroid-based clustering methods that requires a predetermined number of clusters as a parameter for the clustering algorithm. Choosing such a parameter requires a good domain knowledge. In common cases the parameter can be determined by heuristic methods, which may not be accurate enough.

There are related works on Machine Learning and Deep Learning approaches for log analysis as well. Studiawan et al. [9] presented a tool for automatic log parsing based on named entity recognition problem and bidirectional long short-term memory neural network (BLSTM). Meng et al. [10] proposed a tool for anomalies detection in unstructured logs based on long short-term memory neural network as well. Moreover, they presented their own word representation method template2vec inspired by word2vec. First of all the log templates are extracted from historical logs, then template2vec method is applied on obtained templates to get vectors, and finally these vectors are used in trained LSTM neural network, which determines whether a log entry is anomalous. Machine Learning and Deep Learning methods can be powerful, but first they must to be trained before application, thus, there is a need to collect a qualified training dataset. That is a limitation in some cases.

2.2 State-of-the-art systems

There is a lot of log analytics systems which support collecting, processing, and analyzing of log data. In this section the most commonly used log analytics systems will be described.

2.2.1 Splunk

Splunk is a complex log analytics system based on the client-server model. Splunk Forwarder is used to collect the machine generated data from client side and forward to Splunk server. Splunk is centralized logs analysis tool for machine generated data, unstructured/structured and complex multi-line data which provides the following features such as Easy Search/Navigate, Real-Time Visibility, Historical Analytics, Reports, Alerts, Dashboards and Visualization [11].

Advantages:

- Analyzes the aggregate of logs from a big service cluster,
- Finds real-time logs and with faster speed,
2.2. State-of-the-art systems

- Generates report and alerts for the desired search,
- Provides enhanced GUI and real-time visibility in dashboard in various formats,
- Provides quick results by reducing the time to troubleshoot and resolve issues,
- Works like a monitoring, reporting and analysis tool and provides insights,
- Does not require other dependent services (like database),
- Requires minimum hardware resources,
- Easy to setup and low-cost maintenance,
- Accepts any data type including CSV, JSON log formats etc.,
- Monitors Amazon Web Services infrastructure,
- Uploads and indexes log data from a local PC to Splunk directly.

Disadvantages:

- It is an expensive proprietary solution [12].

2.2.2 ELK Stack

ELK is an abbreviation for Elasticsearch, Logstash, Kibana. It is a complex log analytics system consisting of the following three main platforms.

- Elasticsearch - a distributed, RESTful powerful and fast search and analytics engine provided with query language (In fact it is a NoSQL database based on Lucene search engine).
- Logstash - an open-source data collection engine with real-time pipelining capabilities. Logstash can dynamically collect and unify data from disparate sources and normalize the data. Also Logstash comes with lots of plugins, that allow to enrich and expand the functionality for simplifying the ingestion and analysis processes.
- Kibana - a visualisation platform allows to build graphs, histograms and other useful visual data interpretations and dashboards [13].

Advantages:

- Open-source,
- Easy configuration,
- ELK is a total log-analysis platform for search, analyses and visualization of log-generated data from different machines,
2. State-of-the-art

- ELK can securely pull, analyze and visualize data, in real time, from any source and format,

- ELK can perform centralized logging to help identify any server and application-related issues across multiple servers and correlate the logs in a particular time frame,

- Offers plugins to connect with various types of input sources and platforms,

- ELK is geared to handle big data to provide crucial business insights [14, 15].

Disadvantages:
- Different components can become difficult to handle the complex configuration is used,
- Requires Java to be installed on every node since Logstash is written in JRuby [16].

2.2.3 Logalyze

Logalyze is an open source platform providing centralized log management and analysis solution, which supports collecting, parsing, indexing and storing data from any device, application or OS.

Advantages:
- Open-source,
- High rate of data processing,
- Parse any log row with built in or custom made Log Templates,
- Ability to analyze custom business application logs,
- Browse or search logs with a web based administration GUI like with Google,
- Create multidimensional statistics real-time based on individual fields of log,
- Securely transport log data to other LOGalyze engines or syslog devices,
- Export reports or lists into CSV, XLS, PDF or HTML,
- Alert and notify users or other systems when an event matching one or more specified criteria is generated,
- Compatible with rsyslog, syslog-ng, Lasso, Snare,
2.2. State-of-the-art systems

- Connect remotely to SOAP API service [17].

Disadvantages:

- Logs collected in Logalyze own database occupy a significant amount of disk space [16].

2.2.4 Graylog

Another open source centralized system for log analysis. Graylog captures, stores, and enables real-time search and analysis against terabytes of machine data from any component in the IT infrastructure.

Advantages:

- Open source,
- Scalable Log Collection,
- Log Data Enrichment,
- Graphical Log Analysis,
- Alerts and Triggers,
- REST API,
- Extended log collection using “sidecar” (configurations).

Disadvantages:

- Some users notice unintuitive approaches in configuration and management aspects,
- Absence of some useful features in free open source version such as Correlation Engine and Data Forwarder forces to buy full Enterprise edition, which is enough expensive [18].

2.2.5 SolarWind Loggly

Loggly is a cloud-based (SaaS) log analysis tool, that is distributed with subscription model. Since it is a SaaS application, it does not require installation and maintenance on machines.

Advantages:

- Uses unified and standardized format of log files to be stored, that makes the further log analysis easier,
- Easy configuration,

\(^1\)SaaS – System as a Service
2. **State-of-the-art**

- Easy-to-use - there is no need to hire a certified administrator,
- Absence of proprietary query language, uses full-text search and searches by fields instead,
- Uses open protocols to collect data and REST API enables the application to send events directly to Loggly using HTTP or HTTPS/TLS.

Disadvantages:

- Highly dependent on network connectivity, because of its SaaS nature,
- Subscription model is not fine-grained in terms of proportion - price/need,
- Third-party organizations may have an access to private logs, because of its cloud-based nature.
WebLogic log structure

Since technically logs are text streams or files it means that task of log processing is from text retrieval field. Text retrieval is a branch of information retrieval in which the information is stored primarily in the form of text [19].

Log files gathered from different infrastructure components can significantly vary in formatting and semantics, but they still contain markers intended to separate individual elements within the data. Therefore, logs can be considered as semi-structured data [20, 21].

Nowadays, many organizations commonly use SOA as an architectural approach in which applications make use of services available in the network [22]. SOA applications typically use Enterprise Service Bus (ESB) as an integration middleware connecting front-end and back-end applications. It performs transformations of data, ensures messaging and routing, manages composition of multiple requests. The ESB is typically implemented using a specially designed integration runtime that ensures the best possible productivity [23].

ESB often uses WebLogic application servers. In this work the logs from WebLogic server are considered.

The example of structure of the log message format is presented in Figure 3.1.

The description of each element of WebLogic Server log format is listed in table 3.1.

WebLogic Server subsystems or application code send log requests to Logger objects. These Logger objects allocate LogRecord objects which are passed to Handler objects for publication. Both loggers and handlers use severity lev-

\[ \text{<Timestamp><Severity><Subsystem><MessageID><MessageText>} \]

Figure 3.1: WebLogic server log entry pattern
els and (optionally) filters to determine if they are interested in a particular LogRecord object. When it is necessary to publish a LogRecord object externally, a handler can (optionally) use a formatter to localize and format the log message before publishing it to an I/O stream.

Figure 3.2 shows the WebLogic Server logging process. WebLogic Server subsystem or J2EE application invokes a method on one of logging implementations which distributes generated messages to the server Logger object. The server Logger object publishes the generated messages to any message handler that has subscribed to the Logger.

As it was described in Table 3.1 every log entry has a severity level attribute that indicates the potential impact of the event or condition that the message reports. All possible message severity levels are listed in table 3.2 arranged in ascending order.

In this thesis the most interesting and important severity levels for further analysis and completion of defined tasks are warning and higher, but all severity levels are considered during the parsing phase.

As it was described above log entries matches the pattern 3.1, thus each

---

2 J2EE – Java Enterprise Edition
Table 3.2: Message severity levels

<table>
<thead>
<tr>
<th>Severity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE</td>
<td>Used for messages from the Diagnostic Action Library. Upon enabling</td>
</tr>
<tr>
<td></td>
<td>diagnostic instrumentation of server and application classes, TRACE</td>
</tr>
<tr>
<td></td>
<td>messages follow the request path of a method.</td>
</tr>
</tbody>
</table>
| INFO      | Used for reporting normal operations; a low-level informational message.
| NOTICE    | An informational message with a higher level of importance.            |
| WARNING   | A suspicious operation or configuration has occurred but it might not  |
|           | affect normal operation.                                               |
| ERROR     | A user error has occurred. The system or application can handle the    |
|           | error with no interruption and limited degradation of service.         |
| CRITICAL  | A system or service error has occurred. The system can recover but there|
|           | might be a momentary loss or permanent degradation of service.         |
| ALERT     | A particular service is in an unusable state while other parts of the  |
|           | system continue to function. Automatic recovery is not possible; the   |
|           | immediate attention of the administrator is needed to resolve the problem.|
| EMERGENCY | The server is in an unusable state. This severity indicates a severe    |
|           | system failure or panic.                                               |

Figure 3.2: WebLogic Server logging process
3. **WebLogic Log Structure**

Log entry could be easily extracted from the log file and stored to proper object structure that could be conveniently processed in the future.
Vectorization of Log Entries

After we have defined the structure of the log files, it is time to move on to vectorization and the further analysis. Once logs are parsed and stored as structured objects in memory they needed to be transformed into numeric vector representation. This process is called feature extraction or vectorization. It is an essential first step in text analysis, and log analysis can be considered as a narrower field of text analysis.

Representing log messages numerically gives the ability to perform a meaningful analytics and compare vectors to each other. Each vector component is a feature representing attributes and properties of the log message. The obtained vectors can be considered as points in high-dimensional semantic space. Points in space can be close together or far apart, they may be tightly clustered or, on the contrary, evenly distributed [25].

In this chapter a number of vectorization techniques, which are used in this work, will be described.

4.1 Natural Language Processing

Natural Language Processing (NLP) – the branch of computer science and artificial intelligence that gives the computers the ability to understand texts and spoken words and derive meaning from human languages [26, 27]. There are several tasks that are typical for NLP:

- **Speech recognition** – is a task of converting voice data to text data. Speech recognition is required for any application that follows voice commands or answers users’ questions. The main difficulties in this task are fast talking, different accents and intonations, variety of slang,

- **Part of speech tagging**, or grammatical tagging, is the process of determining the part of speech of a particular word or piece of text based on its use and context,
4. **Vectorization of Log Entries**

- **Solving the problem of word sense disambiguation** – determining the most appropriate meaning of the word with multiple meanings depending on the context [27].

- **Named entity recognition**, or NER, identifies and locates words or phrases as entities of real world such as person names, organizations, locations and so on [28].

- **Coreference resolution** is the task of finding expressions that refer to the same entity in a text. It is an important task needed to be solved before proceeding to the next analysis phase such as document summarizing, question answering, or information extraction [29].

- **Sentiment analysis** is a technique that is used to attempts to extract sentiment – positive, negative, or neutral – from given text [30].

- **Natural language generation** is the task of putting structured information into human language. This task can be considered as an opposite task to speech recognition [27].

There is a number of tools and approaches for NLP purposes. For instance, for Python programming language there is one of the most commonly used libraries in NLP field – Natural Language Toolkit, or just NLTK. More information about the library can be found on [31]. In this work NLTK library is used for NLP analysis method of log clustering.

### 4.1.1 Linguistic vectorization approach

To analyze the log data, firstly, the raw text of log messages must be properly preprocessed. The common preprocessing process when solving NLP problems has the workflow as it is shown on Figure 4.1.

![Figure 4.1: NLP Workflow](image)

Once the enumerated steps are done, every log entry from log file is represented as a vector. In the following sections each part of this workflow will be described in details.

### 4.1.2 Tokenization

*Tokenization* in common text processing meaning is a process of splitting the document into subsequences of characters that are grouped together as a useful
4.1. Natural Language Processing

semantic unit of processing. Such subsequences of characters are called tokens. There are some options, how to split the text into tokens. The common practice in NLP area is word tokenization of text data. Thus, applying this approach to log entries there will be a set of vectors consisting of individual words obtained from log entries’ message part.

4.1.3 Dimensionality reduction and normalization

Due to the logs’ nature, along with human-readable text in their messages, they contain a lot of technical details and redundant information from a semantic and linguistic points of view. So, for example, typical error message written in the log contains a stack trace that helps technical experts to locate the error and determine its root cause. Stack traces always have the information about numbers of rows in source code, where the exception were thrown, often there are paths to files in error’s description, special symbols and so on. But since we are speaking of semantics, such information is not necessary and may be removed from the message since it is more important to understand the meaning of the error. Removing non-meaningful data and getting the tokens to a consistent state is called normalization.

The vast majority of software produces the logs in English language. From the perspective of this language there are lots of commonly used words (such as “is” “the”, “a”, “an”, “in” etc.) that have very little meaning and do not add much information to the text. By removing these words, qualitatively there is no loss of information. On the contrary, there is a focus on more valuable and meaningful information.

Typically, all messages in log entries contain some punctuation. Coming back to stack trace example, it could be seen that there are lots of dots, colons, semicolons etc. in there. Punctuation is an important part of any language, however, it is not interesting for our analysis.

Next step in the preprocessing phase is getting the obtained on previous phase words into consistent normalized state. To achieve this state there are commonly used techniques – stemming and lemmatization.

Stemming is a process of transforming related or similar variants of a word to its base form to which suffix can be attached, as they share the same meaning. Thus, stemming transforms words to its parts that never changes. For example, for word ”waiting” the stem will be ”wait”, for word ”connection” the stemmed version is ”connect”. In common case stemming a word may result in words that are not actual words, for example, for word ”computer” the stem is ”comput”, since there are forms ”computing”, ”computed”, ”computation”.

One of the most famous and commonly used stemming algorithms is Snowball Stemming Algorithm derived from older Porter Stemming Algorithm proposed by Martin Porter. The original Porter algorithm suffered from some issues that were fixed in Snowball. For example, when applying Porter Stem-
Vectorization of Log Entries

4. Vectorization of Log Entries

Algorithm on words "fairly" and "sportingly" the result will be "fairli" and "sportingli" respectively, but the results for Snowball algorithm for these words will be "fair" and "sport". 

Another dimensionality reduction NLP technique is lemmatization. Unlike stemming lemmatization usually refers to return the base or dictionary form of a word with the use of a vocabulary and morphological analysis. Such a base form is called lemma. There is an example given to clearly see the difference between stemming and lemmatization. For the word "saw" stemming algorithm will return just "s", but the result of the lemmatization algorithm will be "see". 

So, when the both dimensionality reduction techniques have been discussed, there is a question when to use each of them, and which one is more relevant for the log analytics task. When the speed is prefered over the linguistics then stemming should be used since lemmatization algorithms scan a corpus of words which is a time consuming task. But when the problem is more linguistic and when it is necessary to preserve language norms, preference should be given to lemmatization. In the problem of log analysis stemming approach is more preferable then lemmatization since log messages are mostly written for technical experts who can read and understand it that is such messages may not preserve the language rules and norms. Moreover, the speed of processing the log messages is very important as well, because in the enterprise infrastructure all elements produce massive amounts of logs that must be processed in a reasonable time.

In this master’s thesis stemming is used as a dimensionality reduction technique.

4.1.4 Encoding

After dimensionality reduction phase it’s time to proceed to conversion arrays of tokens into numerical vectors. Vector Space Model is an algebraic model for encoding text documents as vectors. There is a number of vectorization techniques and the choice of a specific one is largely driven by the problem domain. In this paragraph three commonly used approaches to text vectorization will be presented.

4.1.4.1 Term Frequency Vectorization

The first vectorization approach is based on measurement of terms (tokens) frequencies within a document (or a message in case of log entries). According to this vectorization the resulting vector for every log message will be filled with the frequency of each word appears in the log message. The vectors can be straightly counted or normalized with weighting each word by the total number of words in the message. The example of term frequency vectorization is depicted on page 4.2.
4.1. Natural Language Processing

Figure 4.2: Term Frequency Vectorization

Term Frequency approach is widely used, but it has a drawback. Tokens that occur very frequently could be considered as more "significant" than other ones, which occur less often [25]. That does not always correspond to reality.

4.1.4.2 One-Hot Vectorization

One-Hot vectorization is a vector of boolean values. Each such value only indicates the presence of the token within a message. *True (1)* value stands for presence and *False (0)* stands for absence of the token as it is represented on Figure 4.3.

![Figure 4.3: One-Hot Vectorization](image)

Although, it seems to be a good model since it is very simple and transparent, but it is not convenient, because it does not scale well when a large corpus is given, and it disregards the semantics, thus it is not commonly used in NLP tasks. And it is not suitable for log analytics purposes, because of massive amounts of log data. Thus, vector for each log entry would be extremely highly dimensional and sparse, resulting in increased time and computational complexities.
4. Vectorization of Log Entries

4.1.4.3 Term Frequency - Inverse Document Frequency

The vector representations of log entries’ messages that were considered so far only describe a log entry in a standalone fashion, not taking into account the context of the corpus. A better approach is to consider the relative frequency or rareness of tokens in the log message against their frequency in other log messages. The main idea is that the meaning is most likely encoded in the more rare terms from a log message. For example, in a corpus of error log messages, tokens such as "input", "output", and "stream" appear more frequently in log entries describing IOException while other tokens that appear frequently throughout the corpus, like "exception", "weblogic", and "java", are less important.

Figure 4.4: Term Frequency - Inversed Document Frequency

**Term Frequency – Inverse Document Frequency (TF-IDF)**

The vectorization technique normalizes the frequency of terms (tokens) in a log entry message with respect to the rest of the corpus. This approach emphasizes terms that are very relevant to a specific instance, as shown in figure 4.4, where the term has a higher relevance to this message since it only appears there.

**Term Frequency (TF)** – the number of times a term appears in a message, divided by the total number of terms in that message. The division is necessary since the larger messages likely contain more occurrences of the term, so it must be normalized as shown Formula (4.1).

\[
TF(t, m) = \frac{n_t}{N},
\]

where

- \( t \) – the considering term,
- \( m \) – the message containing the considering term,

\(^3\)IOException – Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations [39].
4.1. Natural Language Processing

- $n_t$ – the number of times term $t$ appeared within the message $m$,
- $N$ – the total number of terms in the message $m$.

**Inverse Document Frequency (IDF)** – measures the importance of the token. It is computed as the logarithm of the number of the messages in the corpus divided by the number of messages where the specific token appears (Formula 4.2).

$$IDF(t, M) = \log \frac{|M|}{|\{m_i \in M | t \in m_i\}|}, \quad (4.2)$$

where
- $t$ – the considering term,
- $M$ – the corpus,
- $|M|$ – the number of messages in the corpus,
- $|\{m_i \in M | t \in m_i\}|$ – the number of messages from the corpus $M$ where the term $t$ appears.

Thus, the resulting **TF-IDF** measure for the term $t$ within the message $m$ is a product of $TF(t, m)$ and $IDF(t, M)$ as shown Formula 4.3.

$$TF-IDF(t, m, M) = TF(t, m) \times IDF(t, M) \quad (4.3)$$

Due to its ability to focus on important terms relevant to the considering message and pay less attention to more frequent terms of the whole corpus, **TF-IDF** is a good choice for vectorization of the log messages.

4.1.5 Summary

In this section, the log entries processing from the principles of natural language processing point of view was described. The transformation of the log messages into token arrays is considered, methods of dimensionality reduction, such as stemming and lemmatization, are presented, and the text vectorization techniques such as Term Frequency Vectorization, One-Hot Vectorization and Term Frequency - Inverse Document Frequency are discussed.

All methods have their advantages and disadvantages. The linguistic vectorization approach workflow for log analysis is the following:

- Word tokenization,
- Removing numbers and stopwords,
- Stemming as a dimensionality reduction technique,
- **TF-IDF** as a vector space model.
4. Vectorization of Log Entries

4.2 N-grams vectorization approach

As it was already mentioned logs are machine-generated texts which are not readable and understandable by a regular user of the system. The previous log analytics approach was based on the assumption that similar log entries have similar semantic structure and preserve the meaning. In this section another approach will be considered which is focused on logs’ machine-generated nature. Instead of word tokenization followed by the dimensionality reduction techniques such as stemming and lemmatization, in this section a vectorization approach based on N-grams will be considered.

4.2.1 N-grams tokenization

In computational linguistics area, an N-gram is an overlapping sequence of n tokens from a given sample of text or speech. It does not matter what the tokens are, it can be phonemes, syllables, letters, words or base pairs according to the application, where the N-grams are used.

Unlike the previous linguistic vectorization approach, where the log messages were tokenized into individual words, in this vectorization method of tokenization will rely on symbols. Depending on the value of the N parameter uni-grams, bi-grams, tri-grams and so on are distinguished. There are two main benefits of using n-grams – implementation simplicity and scalability. With the larger N such a model can store more context information, that in certain cases may improve the results. The examples of uni-grams, bi-grams, and tri-grams tokenizations are presented on Figure 4.5, Figure 4.6, and Figure 4.7 respectively.

![Figure 4.5: Tokenization using uni-grams](image)

4.2.2 Dimensionality reduction and normalization

As for the linguistic vectorization method the N-grams require dimensionality reduction as well. But since this approach is a more "machine" than the previous one, there is no need for stopwords removing, CamelCase notation...
4.2. N-grams vectorization approach

4.2.3 Encoding

Vectorization of log messages can be done similarly to Term Frequency Vectorization that was described in 4.1.4.1 but with N-grams as tokens. Hence, for each N-gram the number of its appearances is computed. On the figure [4.8] the bi-grams vectorization is depicted. The resulting values can be normalized with weighting each N-gram by the total number of N-grams in the message.

4.2.4 Summary

In this thesis bi-grams of symbols are used as tokens. Such a choice is justified by the fact that uni-grams of symbols do not describe the log messages enough precisely, it is just a count of individual symbols, that reliable for further message comparison. Tri-grams, four-grams and so on significantly increase sparsity of resulting vectors since with increasing N-gram the number
4. Vectorization of Log Entries

Figure 4.8: Vectorization using bi-grams

of repetitions of such an N-gram within the log message is reduced. Thus, bi-grams ensures the best coverage of the log messages.
Cluster analysis

Cluster analysis is a helpful analytical technique helping to analyze and process a massive amount of log entries in a reasonable time. Several groups (clusters) of similar log messages will be generated by clustering. This approach allows to significantly reduce the dimension of original log data, and allows to analyze not individual log entries, but groups. It simplifies searching and elimination of problems that have arisen during the server runtime.

In this chapter, two clustering methods that are implemented in this master’s thesis are described. The first clustering method is based on cosine similarity measure. The second method is based on Growing Self-Organizing Map neural network.

5.1 Cosine Similarity based clustering

One of the tasks in this thesis is cluster analysis of log entries. Cluster is a structure containing similar log entries. In order to group log entries into the same cluster, it is necessary to compare vectors with each other to determine how similar they are.

In data analytics area there are two main approaches to vectors comparison, distance and similarity. Distance measures how the two vectors are close to each other in the vector space, while similarity measures how two vectors are similar.

One of the most commonly used similarity measures in clustering analysis is cosine similarity.

Let two vectors $\mathbf{A}$ and $\mathbf{B}$ be given. The angle between them is defined as $\theta$ (Figure 5.1).

Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space (Formula 5.1).

$$sim(\mathbf{A}, \mathbf{B}) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \cdot ||\mathbf{B}||},$$

(5.1)
5. Cluster analysis

Figure 5.1: Angle between two vectors \( \mathbf{A} \) and \( \mathbf{B} \)

where

- \( \mathbf{A}, \mathbf{B} \) – the considering vectors,
- \( ||\mathbf{A}||, ||\mathbf{B}|| \) – lengths of the \( A \) and \( B \) vectors respectively.

If two vectors are identical, then the angle between two vectors is 0, hence, \( \cos(0^\circ) = 1 \), and vice versa when two vectors are completely different, then \( \cos(90^\circ) = 0 \).

The cosine similarity is advantageous because even if the two similar documents are far apart by the distance measure because of the size they could still have a smaller angle between them. The smaller the angle between the vectors, the higher their similarity [42].

Cosine similarity is used in this thesis as a similarity measure for log entries clustering.

5.2 Machine Learning clustering

In our days Machine learning is a rapidly growing area. Many algorithms have emerged based on the principles of machine learning, which are actively used in the development of new services, such as email filtering, digital assistants, product recommendation, practical speech recognition, effective web search, fraud detection, and so on [43].

One of the most interesting and perspective branches of machine learning is artificial neural networks.

Artificial neural networks (ANN) are a subset of machine learning algorithms, which conceptually work similarly to human’s brain. ANN consists of a set of so called "neurons". A neuron actually is a mathematical function
that collects and processes the information according to a specific architecture. Neurons are interconnected, and their interconnection may vary depending on type of ANN and the solving problem[44].

The problem of data cluster analysis can be solved with machine learning techniques as well. In this thesis a machine learning methods for log cluster analysis based on Growing Self-Organizing Map model is proposed.

5.3 Categories of machine learning methods

Machine learning methods typically deal with sets of data and use them to learn for themselves. From the learning point of view there are three main categories that the machine learning methods fall into.

5.3.1 Supervised learning

Supervised machine learning trains itself using labeled datasets. This means that the training dataset for such machine learning algorithms must have inputs and correct outputs (labels). Supervised algorithms are often trained in iterative manner. After each iteration the loss function is calculated, which is used for adjusting the algorithm until the error has been sufficiently minimized [45]. The example of the machine learning algorithm that requires supervised learning approach could be a Multi-Layered Perceptron (MLP), which is used for solving of classification problem. MLP is typically trained with Back-propagation algorithm, which after each forward pass of the input vector through a network calculates the output error and propagates it backwards through the network for adjusting the connections between neurons [46].

Supervised machine learning requires less training data than other machine learning methods and makes training easier because the results of the model can be compared to actual labeled results. But labeling the data is a complex task, and there’s the danger of overfitting, creating a model, which works perfectly for the training data, but that it doesn’t handle variations in new data accurately [47].

5.3.2 Unsupervised learning

Another machine learning algorithms category is unsupervised machine learning. Unlike the supervised learning approach this kind of algorithms don’t require labeled data for training, they extract meaningful features needed to label, sort, and classify the data [48].

Unsupervised models are often used for clustering and dimensionality reduction purposes since they are able to identify patterns and relationships in data that humans would miss. For example, such models can successfully
analyze huge volumes of emails and uncover from them features and patterns that help to detect spam \[47\].

This kind of machine learning algorithms is suitable for reaching the goals of this master’s thesis. One of the most interesting unsupervised learning approaches that is able to perform cluster analysis is an ANN called \textit{Self-Organizing Map (SOM)} proposed by the Finnish professor Teuvo Kohonen \[49\] and its improved version called \textit{Growing Self-Organizing Map}.

5.3.3 Semi-supervised learning

Semi-supervised learning is a combination of previously described two machine learning approaches. During training, it uses a smaller labeled dataset to guide classification and feature extraction from a larger, unlabeled dataset. Semi-supervised learning can solve the problem of having not enough labeled data (or not being able to afford to label enough data) to train a supervised learning algorithm \[47\].

5.4 Self-Organizing Map

5.4.1 Description

The SOM neural network is based on the assumption that systems can be designed to emulate the collective cooperation of the neurons in the human brain. Such a kind of neural networks widely used in data mining, visualization of complex data, image processing, speech recognition, process control, diagnostics in industry and medicine, and natural language processing \[50\]. The main idea of SOM model is to map using the Euclidean distance multi-dimensional input vector to low-dimensional structure of neurons according to their characteristics, features and groups similar data together \[51\].

5.4.2 Euclidean distance

In mathematics, the Euclidean distance between two points \(a\) and \(b\) in two-dimensional space (Figure 5.2) is the length of a line segment between this two point, which can be calculated by the Formula \(5.2\).

\[
d(a, b) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2},
\]

where

- \(a, b\) – the points in two-dimensional space,
- \(a_i, b_j\) – coordinates of respective points.

In general case, for points given by Cartesian coordinates in \(n\)-dimensional Euclidean space, the distance can be calculated by the following Formula \(5.3\).
5.4. Self-Organizing Map

Figure 5.2: Euclidean distance between points $a$ and $b$

\[
d(a, b) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \cdots + (b_j - a_i)^2 + \cdots + (b_n - a_n)^2},
\]

where

- $a, b$ – the points in n-dimensional space,
- $a_i, b_j$ – coordinates of respective points.

### 5.4.3 Structure of the SOM

Typically, SOMs consist of input and output layers. Input layer is represented by high-dimensional input vectors, and the output layer, which is also called “Kohonen” or “SOM” layer, consists of neurons organized into low-dimensional structure. Usually neurons form a two-dimensional grid structure since such an arrangement of neurons provides a better representation of clusters. The number of neurons in the output layer denotes the maximum number of clusters and influences the accuracy and generalization capability of the SOM [50]. In practice, often the number of neurons in the output layer corresponds to ten percent of the number of attribute values in the input dataset, but, however, this rule is more of a recommendation, and does not guarantee the best results [52]. The sufficient number of neurons is strongly dependent on the dataset.

The main characteristic for each neuron is a weight vector. Essentially, SOM computes a distance between vectors from the input layer and its neurons weight vector using Euclidean distance.

The structure of SOM which is partitioned into five clusters according to types of input data is presented in Figure 5.3.

In practice, work with the SOM network can be divided into two stages: training and mapping.
5. Cluster analysis

As it was mentioned above SOM is an unsupervised learning algorithm, hence in the training phase in SOM does not rely on predefined target labels that would guide the process.

Once SOM is trained, the mapping phase begins. During this phase the input vectors go through the network and maps to the low-dimensional structure.

5.4.4 Training algorithm

The goal of SOM training is to cause different areas of the network to respond similarly to certain input patterns. This process is motivated by how visual, auditory or other sensory information is handled in separate parts of the cerebral cortex in the human brain.

The SOM training is based on the three following principles:

- Competition – the neurons of the SOM compete among themselves for the opportunity to best represent the input vector,

- Cooperation – in the SOM the spatial location of a neighborhood of cooperating neurons which share common features and represent inputs with similar properties can be determined,

- Adaptation – during the training process weight vectors of the neurons are adjusted towards the input vectors. Thus, the relevant neurons become more similar to the input dataset. Neurons which have a stronger response to a particular input vectors will have an increased chance of responding to similar input in the future [50].
5.4. Self-Organizing Map

At the beginning the SOM is created with given \( m \) and \( n \) parameters determining the SOM’s grid structure, and then neurons are arranged in its nodes.

The neurons’ weight vectors are initialized with small random values typically form the interval \((0; 1)\). Training of the SOM is an iterative process. At each iteration vectors from the input layer are sequentially presented to the SOM model, and the Euclidean distance between each neuron’s weight vector and input vector is computed. It is important to mention that weight vectors’ dimension must correspond with the dimension of vectors from the input dataset since it must be possible to compute the Euclidean distance between them.

The neuron, whose weight vector is the closest to the considering input vector, hence, the Euclidean distance between these two vectors is the smallest, is called the winner or Best-Matching Unit (BMU) (Formula 5.4).

\[
d(x_t) = \arg \min_v \|x_t - w_v\|, v = 1, 2, \ldots, k, \quad (5.4)
\]

where

- \( x_t \) – the input vector presented to the SOM at the iteration \( t \),
- \( w_j \) – the weight vector of neuron \( v \),
- \( k \) – the number of neurons in the SOM calculated as \( m \cdot n \).

The weights of the BMU and neurons from the BMU neighborhood are adjusted towards the input vector. The magnitude of the change decreases with time and with the grid-distance from the BMU. The weights for a neuron \( v \) are adjusted in accordance with the Formula 5.5.

\[
w_v(t + 1) = w_v(t) + \theta(u, v, t) \cdot \eta(t) \cdot (x_t - w_v(k)) \quad (5.5)
\]

where

- \( v \) – the considering neuron,
- \( u \) – the winner neuron,
- \( t \) – the number of the iteration,
- \( w_v \) – the weight vector of neuron \( v \),
- \( x_t \) – the input vector presented to the SOM at the iteration \( t \),
- \( \theta(u, v, t) \) – the neighbourhood function that determines the distance from the winner neuron \( u \) and the considering neuron \( v \) in the SOM’s grid,
- \( \eta(t) \) – the learning rate function at the iteration \( t \).
The neighborhood function $\theta(u, v, t)$ values lay in the range [0; 1]. The implementation of such a function may vary depending on the problem being solved with the SOM. The most simple implementation of could be the function that returns 1 for all neurons that close enough to the BMU (the distance from the BMU to such neurons in the SOM’s grid does not exceed a predefined threshold value) and 0 to others. But the most commonly used neighborhood function is Gaussian \[53\] (Formula \[5.6\]).

$$
\theta(u, v, t) = \exp(- \frac{d_{uv}^2}{2\sigma^2(t)})
$$

(5.6)

where

- $d_{uv}$ – the lattice distance between the winner neuron $u$ and neuron $v$,
- $\sigma(t)$ – the radius of neighborhood at iteration $t$.

Regardless of the functional form, the neighborhood function shrinks with time. At the beginning of the training phase the neighborhood is broad, and the self-organizing takes place on the global scale. When the neighborhood has shrunk to a little number of neurons, the weights are converging to local estimates.

Another important parameter in weight adjustment function is the learning rate $\eta(t)$. The learning rate affect how much the weight vector will be changed at this iteration $t$. This parameter decreases with time to prevent overfitting of the network\[50\].

On the Figure 5.4 the BMU’s neighborhood is depicted.

![BMU’s neighborhood](image)

Figure 5.4: BMU’s neighborhood

The described processes are repeated for the remaining training data until the weights converge and no noticeable changes in the low-dimensional output layer are observed. The entire set of the resulting weight vectors reflects the distribution of the input data \[52\]. Thus, the whole training algorithm can be summarized as:
1. Initialize a SOM instance consisting of neurons arranged into a grid. All neurons’ weight vectors initialized with random values from the interval $[0; 1]$, 

2. Choose a vector $x_t$ from the input layer and compute the Euclidean distance from this vector to every neuron in the SOM, 

3. Find the BMU, the neuron with minimal Euclidean distance to the considering input vector $x_t$ according to Formula 5.4, 

4. Adjust the weight vectors of the BMU and neurons from its neighborhood with Formula 5.5, 

5. Repeat steps 2 to 5 until no noticeable changes in the low-dimensional output layer are observed [50].

5.4.5 Summary

As each algorithm SOM has its own advantages and disadvantages comparing to other clustering techniques.

Advantages:

• Does not make assumptions regarding the distributions of variables nor do they require independence among variables,

• Are able to solve non-linear problems of high complexity,

• Effectively cope with noisy and missing data, very small dimensional and samples of unlimited size [51],

• Powerful representation and visualization abilities.

Disadvantages:

• Relatively high number of parameters that has to be set,

• Computationally expensive [54],

• It requires a predefined topology of the map – number of nodes must be specified,

• Since the neurons are initialized with random values at the beginning, the results of two different SOMs with the same topology may vary [55].

To address some limitation of classical SOM proposed by Kohonen [49] a lot of variations and improved versions of this neural network exist. One of such an improved version called Growing Self-Organized Map will be described in the following section.
5. Cluster analysis

5.5 Growing Self-Organizing Map

5.5.1 Description

As it was described in the previous section, the SOM is normally used to map vectors from high-dimensional input space to two dimensions. The main question for such a dimensionality reduction technique that determines an applicability of the SOM is how accurately the SOM represents the input space.

The SOM is normally organized as a two-dimensional grid of neurons of predetermined size and number of neurons. Predetermining the structure parameters of the SOM is a significant limitation that affects the quality of the final mapping. Maps with different number of neurons applied on the same dataset produce results that significantly differ from each other. Therefore, to find an optimal structure of the SOM there is a need for conducting a number of experiments on different number of neurons parameter values.

To address such a limitation a dynamic model based on SOM ideas and principals is proposed and called Growing Self-Organizing Map (GSOM) [56].

5.5.2 GSOM structure

GSOM is an unsupervised neural network inspired by SOM, but instead of predetermined static structure it involves a dynamic growing in time model.

GSOM starts with only four neurons which weight vectors initialize with random values from the interval $(0; 1)$ (Figure 5.5). And then, during the training process, the number of neurons grows.

![Figure 5.5: The initial state of GSOM](image)

New neurons can be grown only from boundary neurons. The neuron is a boundary if it has at least one of its immediate neighbor positions free. For example, at the beginning all four neurons are boundary. Each neuron in this network can only have four immediate neighbors. When the neuron is chosen
for growth, new neurons will be generated on all its free neighbor positions as it is depicted in Figure 5.6.

![Figure 5.6: New neurons growth](image)

In order for the network to grow new neurons, the following condition in Formula 5.7 must be satisfied.

\[
TE_i > GT
\]  

(5.7)

where

- \( TE_i \) – total error value accumulated by neuron \( i \), from which a new neuron is going to be generated,
- \( GT \) – predefined growing threshold value.

The new neurons have been grown need their weight vectors to be initialized. Random initialization of weight vectors is not suitable anymore since the neurons with randomly generated weight vectors will not match their neighborhood in this case. Alternatively, the weight vectors of new neurons can be filled with values computed from the weight vectors of their neighboring neurons. There are four cases that need to be considered. In the following cases \( w_1, w_2 \) are weight vectors of two already existed neurons in the GSOM, \( w_{\text{new}} \) is a weight vector of newly generated neuron.

1. Two consecutive neighboring neurons on one of new neuron’s sides (Figure 5.7)

\[
w_{\text{new}} = \begin{cases} 
  w_1 - (w_2 - w_1) & \text{if } w_2 > w_1 \\
  w_1 + (w_1 - w_2) & \text{if } w_2 < w_1
\end{cases}
\]

2. The newly generated neuron is arranged between two neighboring neurons (Figure 5.8)

\[
w_{\text{new}} = \frac{w_1 + w_2}{2}
\]
5. **Cluster analysis**

![Figure 5.7: New neuron generation case 1](image1)

![Figure 5.8: New neuron generation case 2](image2)

In this case the mean value of neighboring weight vectors is assigned to the newly generated neuron’s weight vector.

3. The newly generated neuron has one direct neighboring neuron, and this neighboring neuron has its own neighbor on one side which is not directly opposite to the newly generated neuron (Figure 5.9). This case is the same as case 1. The only difference is the position of the neighbor. In cases when both options are available, case 1 is preferable.

\[
w_{new} = \begin{cases} 
  w_1 - (w_2 - w_1) & \text{if } w_2 > w_1 \\
  w_1 + (w_1 - w_2) & \text{if } w_2 < w_1
\end{cases}
\]

![Figure 5.9: New neuron generation case 3](image3)

4. The newly generated neuron has only one neighboring neuron (Figure 5.10)

\[
w_{new} = \frac{r_1 + r_2}{2}
\]

![Figure 5.10: New neuron generation case 4](image4)

\(r_1, r_2\) are the lower and upper values of the range of the neighboring neuron’s weight vector distribution respectively.
5.5. Growing Self-Organizing Map

Number of neurons in the GSOM grows until the network will cover the training data in the best way.

5.5.3 Training algorithm

The training process is based on the same competition, cooperation and adaptation principles as was in SOM network and consists of three phases.

5.5.3.1 Initialization phase

In this phase a new GSOM model is initialized with four neurons, weight vectors of which are filled with random values from the interval (0; 1). Then the growth threshold (GT) value is calculated by the Formula [5.8]

\[
GT = -D \times \ln(SF),
\]

where

- \( D \) – dimensionality of input data vectors,
- \( SF \) – value of spread factor parameter.

\( SF \) parameter controls the growth of the network. Value of this parameter must belong to the interval (0; 1).

5.5.3.2 Growing phase

1. Choose a vector \( x_t \) from the input layer and compute the Euclidean distance from this vector to every neuron in the GSOM,

2. Find the BMU, the neuron \( v \) with minimal Euclidean distance to the considering input vector \( x_t \) according to formula:

\[
d(x_t) = \arg \min_v ||x_t - w_v||, v = 1, 2, \ldots, k
\]

3. Adjust the weight vectors of the BMU \( u \) and neurons from its neighborhood with formula:

\[
w_v(t + 1) = w_v(t) + \theta(u, v, t) \cdot \eta(t) \cdot (x_t - w_v(k))
\]

4. Decrease the learning rate \( \eta(t) \) with the following formula:

\[
\eta(t + 1) = \alpha(1 - \frac{R}{n(t)})\eta(t)
\]

where \( \alpha \) is reduction constant from the interval (0; 1), \( R = 3.8 \) is a constant chosen since the starting number of neurons is four; \( n(t) \) is a number of neurons at the iteration \( t \),
5. Cluster analysis

5. Increase the total error value \( (TE_i) \) of the BMU by the measured distance to the input vector

\[
TE_i = E_i^u + d(x_i)
\]

6. Grow new neurons if the BMU is a boundary neuron and \( TE_i > GT \) and initialize their weight vectors according to the rules described in section 5.5.2. Distribute the error \( TE_i \) to the neighboring neurons if the BMU in not boundary with the following formulas.

\[
E_{i+1}^u = \frac{GT}{2}
\]

\[
E_{i+1}^i = E_i^i + \gamma E_i^i
\]

\( E_{i+1}^u \) is the error of the BMU reduced to half the growth threshold. \( E_{i+1}^i \) is the error of \( i \)th direct neighbor of the BMU. Increasing of this error is controlled by the factor of distribution constant \( \gamma \) which is selected at the beginning of the training and whose value belongs to the interval \( (0; 1) \). These two equations simulate the spreading effect of the error outwards from the BMU.

7. If new neurons were generated on the previous step, reset the learning rate to its starting value.

8. Repeat steps 1 to 7 until the node growth is reduced to a minimum level.

9. Remove all neuron from the GSOM which never became BMUs to reduce the number of redundant neurons [56].

5.5.3.3 Smoothing phase

In this phase all steps from the previous phase are repeated, but without growing new neurons. The learning rate \( \eta(t) \) is reduced in comparison with growth phase and the neighborhood function is fixed to immediate neighbors.

Smoothing serves as an additional adjustment of weight vectors of the GSOM’s neurons and stops when error values of neurons become very small.

5.5.4 GSOM clustering

Once, the GSOM model is trained it is able to map high-dimensional vectors from the input dataset to a two-dimensional structure. Similar input vectors are mapped close to each other in the grid, forming regions of neurons (clusters). It is necessary to identify these clusters since this information will be used to label the input vectors.

For this purpose K-Means algorithm can be used, which is another well-known and commonly used machine learning clustering technique [57]. The
advantage of using K-Means on network’s neurons rather than on the dataset itself is that number of neurons is significantly less than the number of vectors in the input space. As input vectors for K-Means the weight vectors of neurons are used.

K-Means algorithm provides a partition clustering that uses minimum squared error criterion when grouping the data. The algorithm randomly chooses $k$ cluster centroids in the input data, assigns each input vector to the closest cluster, then for each obtained cluster its centroid recalculated as a mean of all vectors assigned to its cluster. The process is repeated iteratively until the minimum sum of squared distances from each point of the cluster to its centroid is reached [58].

K-Means is sensitive to centroid initialization. Therefore, the algorithm runs several times with different $k$ parameter value from 2 to $\sqrt{N}$, where $N$ is the number of neurons in GSOM [59]. The best cluster partitioning is identified with lowest Davies-Bouldin index (DB), which measures the within-cluster variation and between-cluster variation for the resulting clusters (Formula 5.9).

$$DB = \frac{1}{n} \sum_{i=1, i\neq j}^{n} \max\left(\frac{S_i + S_j}{d(c_i, c_j)}\right),$$

(5.9)

where

- $n$ – number of clusters,
- $S_i, S_j$ – within cluster variations,
- $d(c_i, c_j)$ – between cluster variation.

### 5.5.5 Summary

GSOM preserves the advantages of classical SOM and due to the ability to grow GSOM addresses the limitations with predefined structure and number of neurons. In this master’s thesis GSOM is implemented as an another clustering method.
In this master’s thesis the two approaches to cluster analysis of log data are considered. The first approach – with cosine similarity between log entries. The second – via Growing Self-Organizing Map (GSOM).

Moreover, two vectorization methods are used. The first method is based on NLP ideas of text processing, linguistic vectorization. Such an approach can improve cluster analysis via preserving the semantic structure of the messages. In the second method log messages are primarily considered as a machine-generated text requiring the appropriate processing technique. Vectorization by N-grams is used as a text processing technique that does not take into account the semantics of log messages.

As a part of this master’s thesis, application for the log analytics purposes supporting the described text vectorization techniques and clustering approaches were developed. In this chapter the design of the application and implementation will be described in detail.

6.1 Requirements

Software requirements are divided into two types: functional and non-functional. Functional requirements are a set of functions that the application should provide. Non-functional requirements, in turn, describe the properties of the application and the technical constraints that it must comply with.

6.1.1 Non-functional requirements

- The application should be used on machines running under Linux operation system where WebLogic server deployed,

- The application should be designed as CLI application,
6. **Implementation**

- The application should be developed in Python 3.8 programming language since it is supported on Linux operation systems and has a lot of libraries and frameworks used in data analysis and NLP processing.

6.1.2 **Functional requirements**

- The application should read and parse log data files which is passed as an input,
- The application should support NLP vectorization,
- The application should support N-grams vectorization,
- The application should support clustering technique based on cosine similarity,
- The application should allow users to specify similarity threshold parameter for cosine similarity based clustering,
- The application should support clustering technique based on GSOM,
- The application should allow users to specify all parameters for GSOM clustering,
- The application should be able to save trained GSOM model,
- The application should be able to load a trained GSOM model,
- The application should be able to save found clusters on disk,
- The application should create a report in CSV file,
- The application should be able to save the report on disk.

6.2 **Use cases**

Use case diagrams usually describes a set of actions (use cases) that actors (user of another system) should or can perform [60].

The application is designed as a CLI application and allows user to interact with itself via predefined commands. User can request a helping info that should advise him how to use the application. User can choose between two clustering techniques, cosine similarity and GSOM. He also can specify a number of parameters for each algorithm, decide, whether he wants to save the results of clustering, or trained GSOM model to load it in future. Finally user can choose between N-grams and linguistic vectorization methods. The use case diagram is depicted in Figure 6.1.

---

5 CSV – Comma-Separated Values
6.3 Modules

In this master’s thesis, an application for cluster analysis of server logs has been developed. The application is designed as a group of interconnected components, modules (Figure 6.2). Each step of the workflow is accompanied by modules that are used at this step. In this section, each module will be described in more detail. The application workflow is depicted in the Figure 6.2.
6. Implementation

6.3.1 LogEntry

This module contains only one class LogEntry. Typically, log data is represented as an unstructured or semi-structured text. Before analysis it is necessary to transform it to a structured form. All log entries in the input log data file are parsed and stored as instances of class LogEntry.

6.3.2 Parser

Parser module consists of a Parser class, which is responsible for reading logs, extracting information from them and transform the retrieved information in structured format in LogEntry class instances. Parser opens the input log file and sequentially reads it by lines and finds messages matching the pattern described in the chapter 3. Such messages then are split into separate parts. The first part is always a timestamp. For the convenient ordering of log entries the timestamp of each log entry is parsed as a date. All obtained parts are used for creating LogEntry instances. Then these instances are used in the vectorization and clustering processes.

6.3.3 Vectorization

In this module the process of transformation of log entries messages into vector is performed. There are four classes: Vectorizer, TfVectorizer, TfIdfVectorizer, and NgramVectorizer.

Vectorizer is the base class specifying the interface for other classes in this module. TfVectorizer provides the user with term-frequency vectorization model. The number of occurrences of individual normalized words within the log message is calculated. TfIdfVectorizer applies TF-IDF vectorization approach. Normalization is performed as it was described in section 4.1.1, each log message is cleaned from redundant information such as numbers, punctuation, and stopwords, then the log message is tokenized by words. All tokens written in camel cases notation are parsed into individual words. Then all obtained words are unified with stemming technique.

NgramsVectorizer transforms the input messages of log entries into n-grams of given parameter $n$ and calculates occurrences of overlapping sequences of characters, n-grams. In this work bi-grams are considered that means that parameter $n = 2$.

6.3.4 Clustering

This module deals with everything related to clustering in this thesis. Here a cluster structure is defined as a class Cluster. All similar log entries should fall into the same cluster. Moreover, server produces massive amounts of log entries, several dozens or even hundreds of log entries can be produces in a second. For more convenient analysis Cluster class stores log entries on its
timestamp into a dictionary structure. If several log entries with the same timestamp should be grouped into the same cluster they stored as a list under this timestamp in the dictionary. Each cluster is characterized by its reference log entry and label.

In Clustering class the cosine similarity clustering approach is implemented. Log entries sequentially goes through the chosen vectorizer, and the obtained vectors are compared with reference log entries of already existing clusters. The considering log entry is added to the cluster if the cosine of an angle between vector of this log entry and vector of the reference log entry of the considering cluster is greater than the threshold predefined by user. When no appropriate cluster had not been found, a new cluster is created and the considering log entry is assigned to this cluster as its reference log entry.

GSOMClustering class provides clustering technique based on GSOM neural network. Similarly to the Clustering class at the beginning log entries goes through the chosen vectorizer. Then, according to GSOM nature the BMU, or winner neuron, is defined. After training of the GSOM each neuron is labeled with cluster label to which this neuron belongs to. Thus, the label obtained from the BMU neuron is exactly the cluster to which the considering log entry will be added. The same as in the case of cosine similarity clustering, when there is no appropriate cluster, a new one is created with the considering log entry as a reference log entry.

6.3.5 Neural

Module Neural contains classes Neuron and GrowingSelfOrganizingMap.

Neuron is a very important class since it is a base unit of the GSOM. Every GSOM’s neuron is characterized by its weight vector which dimensionality depends on the dimensionality of the input vectors. In the GSOM architecture proposed in this master’s thesis neurons have references on their direct neighbors (left, top, right, and bottom). Such an approach simplifies determining the appropriate growing strategy for GSOM and updating the weight vectors within the neighborhood during the training phase. Each neuron handles its error value and number of “hits”, number of times when the neuron had been chosen as a BMU. Information about neurons’ coordinates and cluster label to which the neurons belong to are handled as well. Neurons before training generate their weight vectors with random values form the interval (0; 1). During the training phase neurons adjust their weight vectors towards the input vectors.

GrowingSelfOrganizingMap deals with creating, configuring, training and using the GSOM neural network. At the beginning the GSOM is initialized with four connected neurons. Then, during the training phase this number grows until achieving the best covering of the input data. The growing rules and training algorithm are described in section 5.5 in detail. The Euclidean distance between vectors from the training dataset to every existing neuron
in the neural network is computed. Then, the neuron with the minimal distance is chosen as a BMU, or the winner. The weight vectors of the BMU and the neurons from its neighborhood are adjusted towards the considering training vector. If the BMU is a boundary neuron, new neurons on all its free neighboring positions are generated and initialized with the weights according to the appropriate weight generating strategy, and the learning rate is reset to its starting value. If the BMU is not boundary, the error is distributed among the neurons neighboring to the BMU. The training phase is limited by the number of so called epochs. During one epoch, the entire training dataset goes through the neural network.

Because of the fact that after training there are a number of neurons that had never been BMUs, they number of "hits" is equal to zero. Such neurons are called "dummy" neurons. They don’t respond to any input vector and, then, can be removed.

Once the GSOM is trained and cleaned from "dummy" neurons, the smoothing phase is started. No new neurons are generated in this phase, only the weight vectors of existing neurons are adjusted. In contrast to weight vectors adjustment in the training phase, the only immediate neighbors of the BMUs are adjusted.

During all these phases the neurons are partitioned into groups that further will define the clusters to that the GSOM is able to cluster log entries. Partitioning of the neurons is performed with K-Means algorithm that iteratively runs of neurons’ weight vectors. The quality of grouping is managed by DB index. The best groups partitioning has the lowest DB index.

6.3.6 Reporting

After the log data is partitioned into clusters using any of the implemented algorithms, the report containing the clustering results is generated. Reports are stored as CVS files into "reports" folder in the root folder of the application. The example of the content of such reports is shown in Listing 6.1

```
TIMESTAMP, CLUSTER_LABEL, INFO, WARNING, ERROR, CRITICAL
2021-03-04 16:41:33+00:00, Cluster 1, 0, 0, 5
2021-03-04 16:41:33+00:00, Cluster 2, 0, 0, 5
2021-03-04 16:40:56+00:00, Cluster 3, 0, 10, 0
```

Listing 6.1: Clustering report example

Detailed description of each cluster containing the reference log entries of each cluster is stored by reporting module in CSV format as well. The example of such a file is shown in Listing 6.2

```
CLUSTER_LABEL, CLUSTER_REF
Cluster 1, ExceptionexecOpcode
Cluster 2, Caught Exception calling an opcodeError calling opcode
          PCMOPARGETACCTBILLS exception execOpcode
Cluster 3, Invoking service request locally failed
          oraclefabriccommonBusinessFaultException
```

46
Listing 6.2: Information about clusters generated by reportin module example

Reporting module can store the information about the GSOM training process as a CSV file (Listing 6.3).

<table>
<thead>
<tr>
<th>ITERATION</th>
<th>PHASE</th>
<th>DB_INDEX</th>
<th>CLUSTERS_NUMBER</th>
<th>NEURONS_NUMBER</th>
<th>SATURATION</th>
<th>QUANTIZATION_ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>training</td>
<td>0.6221479345961244</td>
<td>14, 481, 477</td>
<td>0.1523827162441621</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>training</td>
<td>0.4127429311822514</td>
<td>18, 725, 244</td>
<td>0.1321224019310347</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>training</td>
<td>0.5413733490333446</td>
<td>2, 927, 202</td>
<td>0.128492611218986</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>training</td>
<td>0.639091023859594</td>
<td>28, 1079, 152</td>
<td>0.12350828591288247</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>training</td>
<td>0.6775581125752507</td>
<td>31, 1192, 113</td>
<td>0.11286353087248327</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>training</td>
<td>0.5961215920526367</td>
<td>34, 1282, 90</td>
<td>0.10763564648018721</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>training</td>
<td>0.6263253326664563</td>
<td>35, 1343, 61</td>
<td>0.10436261511541332</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>training</td>
<td>0.623689576939242</td>
<td>33, 1407, 64</td>
<td>0.10010499026297093</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>training</td>
<td>0.6462190955448166</td>
<td>31, 1467, 60</td>
<td>0.09575238847989083</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>training</td>
<td>0.6702914606133557</td>
<td>31, 1521, 54</td>
<td>0.09251777708547009</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Listing 6.3: GSOM training report example

6.3.7 Metrics

In this module there are cosine similarity and euclidean distance algorithms are implemented. The implementation of cosine similarity algorithm is presented in Listing 6.4.

```python
def cosine_similarity(vector_1, vector_2):
    sum_ = 0
    smaller = vector_1
    larger = vector_2
    if len(smaller) > len(larger):
        smaller = vector_2
        larger = vector_1
    for key in smaller.keys():
        larger_value = larger.get(key)
        if not larger_value:
            continue
        sum_ += float(smaller[key]) * float(larger[key])
    a = __vector_length(smaller)
    b = __vector_length(larger)
    similarity = float("{:.7f}".format(sum_ / (a * b)))
    return similarity
```

Listing 6.4: Cosine similarity algorithm

The implementation of Euclidean distance is represented in Listing 6.5.

```python
def euclidean_distance(vector_1, vector_2):
    sum_ = 0
    for key, v1_value in vector_1.items():
```
6. Implementation

```python
v2_value = vector_2.get(key)
if not v2_value:
v2_value = 0
sum_ += (float(v1_value) - float(v2_value))**2
distance = float("{:.7f}".format(sqrt(sum_)))
return distance
```

Listing 6.5: Euclidean distance algorithm

6.3.8 Evaluator

Evaluator module deals with evaluation of clustering approaches implemented in this master’s thesis. There is Evaluator class in this module that is used for evaluation of results of clustering methods. First of all, a confusion matrix is computed. Then using the values from this matrix such evaluation metrics as Precision, Recall and Adjusted Rand Index are calculated.

6.3.9 Main

The Main module is the connection link for all other modules. This is the place where the calls of the methods of the described classes are called, forming the application workflow. In this module a command line interface allowing user to interact with the application is implemented.

6.4 Command Line Interface

The application is designed as a command line application since the interaction with the server on which this application is performed via the command line. The Command Line Interface (CLI), is a non-graphical, text-based interface to the computer system, where the user types in a command and the computer then successfully executes it. The CLI terminal accepts the commands that the user types and passes to a shell. In the Listing 6.6 the output of the application after running help command is presented.

```
$ python analyzer/main.py --help
Usage: main.py [OPTIONS] COMMAND [ARGS]...
Log Analysis Tool. Parses and clusters log files. Reports are saved to 'reports' folder.

Options:
--version    Show the version and exit.
--help       Show this message and exit.

Commands:
    cos           Runs clustering based on cosine similarity.
    gsom          Runs GSOM clustering.
    method-eval   Runs evaluation of the chosen model.
```

Listing 6.6: CLI for helping info about the developed application for log clustering
Listing 6.7 shows how the helping information for N-grams approach looks like.

```
$ python analyzer/main.py cos --help
Usage: main.py cos [OPTIONS] SOURCE...

Runs clustering based on cosine similarity.
Options:
-V, --vectorization STRING  Vectorization type (ngrams or tf-idf).
                           [default: ngrams]
-n NUMBER  Ngrams dimensionality.  [default: 2]
-t, --treshold REAL  Similarity treshold value.  [default: 0.995]
-S, --save  Save results of clustering to 'clusters' folder.
--help  Show this message and exit.
```

Listing 6.7: CLI for helping info about Cosine similarity based clustering

Listing 6.8 represents the helping information about GSOM.

```
$ python analyzer/main.py gsom --help
Usage: main.py gsom [OPTIONS] SOURCE...
Or
path(s) to prepared log entries .pickle file(s).

Runs GSOM clustering.  [SOURCE] - 1 or more paths to log file(s).
Or
path(s) to prepared log entries .pickle file(s).

Options:
-sf, --spread-factor REAL  Spread factor.  [default: 0.83]
-smf, --smooth-factor REAL  Smoothing factor.  [default: 0.8]
-a, --alpha REAL  Alpha.  [default: 0.9]
-lr, --learning-rate REAL  Learning rate.  [default: 0.3]
-fd, --distribution REAL  Factor of distribution.  [default: 0.3]
-nb, --neighbourhood NUMBER  Max neighbourhood around the winner neuron.
                           [default: 6]
-r REAL  R parameter.  [default: 3.8]
-gs, --gsom PATH  Path to existing trained GSOM.
-t, --training PATH  Path to training data.
-te, --training-epochs NUMBER  Total numbers of epochs for training.
                           [default: 150]
-se, --smoothing-epochs NUMBER  Total numbers of epochs for smoothing.
                           [default: 150]
-S, --save  Save results of clustering to 'clusters' folder.
-SM, --save-model STRING  Save trained GSOM model to a file with
                           given name.  Timestamp will be added to the name.
                           Example: gsom_<timestamp>.pickle  [default: gsom]
-V, --vectorization STRING  Vectorization type (ngrams or tf-idf).
                           [default: ngrams]
-vf, --vectors_file PATH  Path to vectors file.  It is required for
                           TF-IDF vectorization.
```
6. Implementation

| -n NUMBER | Ngrams dimensionality. [default: 1] |
| --help    | Show this message and exit. |

Listing 6.8: CLI for helping info about GSOM based clustering

The work of the developed application is presented in more detail in the Appendix [D].
In this chapter the experiments on large server logs will be discussed.

In this thesis three real WebLogic server instances running in the same cluster were considered as a domain for the log analytics tasks. The fact that server instances share the same cluster means they contain the same copy of applications and objects [62]. All logs are collected for the period from 24.02.2021 to 04.03.2021.

Evaluation of log clustering methods implemented in this thesis were performed, and the experiments with implemented clustering approaches were conducted. The results will be presented in this chapter.

### 7.1 Evaluation

Before applying any method on practice, it is necessary to evaluate it. For the evaluation purposes a subset consisting of 120 log entries was manually gathered from the logs of the mentioned period. This subset were partitioned into 12 clusters as it is shown in Figure 7.1.

![Clustered evaluation dataset](image.png)

Figure 7.1: Clustered evaluation dataset
7. Evaluation and Experiments

7.1.1 Rand Index

The Rand Index in statistics, and in particular in data clustering, is a measure of the similarity between two data clusterings. Rand index is related to the accuracy, but is applicable even when class labels are not used. Mathematically Rand Index can be defined in the following way.

Given a set \( S \) of \( n \) elements \( S = \{e_1, \ldots, e_n\} \) and two partitions of \( S \) to compare. The first partition \( X = \{X_1, \ldots, X_k\} \) into \( k \) clusters, and the second partition \( Y = \{Y_1, \ldots, Y_l\} \) into \( l \) clusters. Define the following statements:

- \( a \) – the number of pairs of elements in the original set \( S \) that are in the same cluster in \( X \) and in the same cluster in \( Y \),
- \( b \) – the number of pairs of elements in the original set \( S \) that are in different clusters in \( X \) and in the same cluster in \( Y \),
- \( c \) – the number of pairs of elements in the original set \( S \) that are in the same cluster in \( X \) and in different clusters in \( Y \),
- \( d \) – the number of pairs of elements in the original set \( S \) that are in different clusters in \( X \) and in different clusters in \( Y \).

Hence, the Rand Index, \( RI \) can be calculated as shown in Formula 7.1:

\[
RI = \frac{a + b}{a + b + c + d} \quad (7.1)
\]

The Rand index is often close to 1.0 even if the clusterings themselves differ significantly. In practice there often is a majority of element pairs that are assigned the different pair label under both the predicted and the ground truth clustering resulting in a high proportion of pair labels that agree, which leads subsequently to a high score\[64\]. Thus, the adjusted version of Random Index is commonly used in practice (Formula 7.2). The adjusted Rand index is ensured to have a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical (up to a permutation)\[65\].

\[
ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \quad (7.2)
\]

where \( E[RI] \) – the expected \( RI \).

Easy to see that described above variables \( a, b, c, d \) in their meaning correspond with True Positive (TP), False Positive (FP), False Negative (FN), True Negative (TN) measures respectively, that are commonly used in Information Retrieval area. And the Rand Index itself can be considered as accuracy of clustering over the pairs of elements in \( S \).
7.1. Evaluation

These measures are used in classification tasks for comparing the results of the classifier that is being examined on some dataset with ground truth labels that are known for this dataset [64]. For clustering analysis, where ground truth labels, typically, are not known, these metrics can be used as well since there is the evaluation dataset, that was manually gathered and clustered from the server logs and could be considered as the ground truth.

Based on these measurements, a confusion matrix, presented on figure 7.2 is built.

![Confusion matrix](image)

Figure 7.2: Confusion matrix

Using $TP$, $FP$, $FN$, $TN$ precision (Formula 7.3) and recall (Formula 7.4) metrics can be computed:

$$precision = \frac{TP}{TP + FP} \quad \text{(7.3)}$$

$$recall = \frac{TP}{TP + FN} \quad \text{(7.4)}$$

These metrics will be useful when comparing different log analytics methods.

Precision can be interpreted as the ratio of pairs of log entries that were correctly grouped into the same cluster to the total number of pairs of log entries that were grouped into the same cluster by the algorithm. While recall is the ratio of pairs of log entries that were correctly grouped into the same cluster to the number of pairs log entries that were correctly grouped into the same cluster and the pairs log entries that were wrongly distributed among different clusters by the algorithm.

Knowing the precision and recall, it is possible to calculate the combined metric $F$-measure which can be considered as a measure of model’s accuracy (Formula 7.5).

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad \text{(7.5)}$$
Thus in this thesis *Adjusted Rand Index*, *precision*, *recall*, and *F-measure* are chosen as clustering evaluation metrics.

## 7.2 Implemented clustering approaches

The application implemented in this master’s thesis allows to cluster logs using two different methods. The first method is based on cosine similarity, and the second is based on the GSOM neural network. Each of these methods can use both linguistic and N-grams vectorization approaches. Thus, there are four combinations of clustering methods with vectorization techniques:

- Cosine similarity based clustering with Linguistic vectorization,
- Cosine similarity based clustering with N-grams vectorization,
- GSOM clustering with Linguistic vectorization,
- GSOM clustering with N-grams vectorization.

All enumerated clustering approaches are implemented in this thesis and applied on evaluation dataset constructed from log data of the real WebLogic server.

### 7.2.1 Cosine similarity based clustering with Linguistic vectorization

The first method that was applied on evaluation dataset is cosine similarity based clustering with linguistic vectorization. All log entries from the evaluation dataset are preprocessed with NLP techniques and transformed into vectors with TF-IDF vectorization. Then, the obtained vectors are partitioned into clusters with cosine similarity. The similarity threshold is set at 0.995 since it is required to define clusters as precisely as it possible.

20 clusters have been found by this approach, while the evaluation set consists of the log entries belonging to only 12 clusters. The log entries distribution among clusters obtained using the linguistic vectorization method is shown in Figure 7.3.

In the Table 7.1 the evaluation results of this approach are presented.

Relatively low recall score can be explained by the greater than it was expected number of found clusters. An increase in the number of clusters provoked an increase in the number of false negatives for log entries pairs which should be in the same cluster, but they were partitioned into separate clusters.

Nevertheless, the precision score is high. That means the number of log entries pairs that were falsely grouped into the same cluster is low in comparison to the number of pair that were correctly grouped into the same cluster.
7.2. Implemented clustering approaches

Figure 7.3: Cosine similarity based clustering with linguistic vectorization

Table 7.1: Cosine similarity based clustering with linguistic vectorization

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total clusters found</td>
<td>20</td>
</tr>
<tr>
<td>Precision</td>
<td>0.97</td>
</tr>
<tr>
<td>Recall</td>
<td>0.76</td>
</tr>
<tr>
<td>ARI</td>
<td>0.83</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.85</td>
</tr>
</tbody>
</table>

The overall accuracy of the algorithm is reflected in the ARI score that is equal to 0.83 and in F-measure that is equal to 0.85.

7.2.2 Cosine similarity based clustering with N-grams vectorization

The second applied clustering method is cosine similarity based clustering with N-grams vectorization. Instead of TF-IDF vectorization technique the N-grams are used. The length of N-gram token is two symbols, thus this clustering method deals with bi-grams vectorization. Bi-grams vectorization ensures the best coverage of log messages. The similarity threshold is set at 0.995 as in the previous approach.

In comparison with the cosine similarity based clustering with linguistic approach this method partitioned the evaluation dataset to 19 clusters. This result, though not significantly, but closer to the target. The log entries distribution among clusters using cosine similarity based clustering with N-grams vectorization is depicted in Figure 7.4.

Evaluation results are presented in the Table 7.2.

It can be seen that all characteristics have increased in comparison with the linguistic vectorization. The precision of cosine similarity based clustering
7. Evaluation and Experiments

Figure 7.4: Cosine similarity based clustering with N-grams vectorization

Table 7.2: Cosine similarity based clustering with N-grams vectorization evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total clusters found</td>
<td>19</td>
</tr>
<tr>
<td>Precision</td>
<td>0.98</td>
</tr>
<tr>
<td>Recall</td>
<td>0.78</td>
</tr>
<tr>
<td>ARI</td>
<td>0.84</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.87</td>
</tr>
</tbody>
</table>

with N-grams vectorization is 0.98 against 0.97 for the linguistic method, recall 0.78 against 0.76, ARI 0.84 against 0.83, and F-measure 0.87 against 0.85.

Such evaluation results tell that the N-grams vectorization method which does not rely on semantics, but works with log entries like with machine-generated text, shows better results and is a better choice when cosine similarity based clustering analysis is being performed on log data.

7.2.3 GSOM clustering with Linguistic vectorization

Another clustering approach is GSOM. In case of GSOM it is possible to use either linguistic vectorization or N-grams vectorization as well. In this subsection the outcomes of GSOM clustering with linguistic vectorization will be presented.

The GSOM with linguistic vectorization were trained on the training dataset obtained via combining log entries from several log files of the server. The number of log entries in the training dataset is 36257.

Initially, network training was run with 200 iterations for the training phase and 50 smoothing iterations. New neurons stopped generating after iteration 143, which means that after this iteration the network reached sat-
7.2. Implemented clustering approaches

The quantization error stopped decreasing significantly after 150 iterations. The graph of the change in the quantization error during training of the GSOM network is shown in Figure 7.5.

![Figure 7.5: GSOM with linguistic vectorization quantization error graph](image)

After training, the network consisted of 1885 neurons. The minimum value of the DB index was equal to 0.39, which corresponded to 37 clusters, which the network is able to determine. The trained neural network is depicted in Figure 7.6. Different colors correspond to different clusters.

![Figure 7.6: Trained GSOM with linguistic vectorization](image)

Applying the GSOM with linguistic vectorization on the evaluation dataset it can be seen that unlike the previous methods GSOM with linguistic vector-
7. Evaluation and Experiments

ization has partitioned the dataset into 11 clusters. This result is more closer to the target number of clusters. The diagram of log entries distribution among clusters is shown in Figure 7.7.

![Figure 7.7: GSOM with linguistic vectorization clustering](image)

Evaluation metrics are collected in the Table 7.3.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total clusters found</td>
<td>11</td>
</tr>
<tr>
<td>Precision</td>
<td>0.83</td>
</tr>
<tr>
<td>Recall</td>
<td>0.95</td>
</tr>
<tr>
<td>ARI</td>
<td>0.84</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.88</td>
</tr>
</tbody>
</table>

In comparison with both previous method, GSOM with linguistic vectorization shows significantly better results in recall, 0.95 against 0.76 and 0.78 for cosine similarity based clustering with linguistic and N-grams vectorizations respectively. Precision score, on the contrary, have been decreased against the previous approaches, where the precision score was 0.97 and 0.98 respectively. Nevertheless, the ARI score of GSOM with linguistic vectorization is 0.84 that is higher than 0.83 for cosine similarity based clustering with linguistic. The F-measure score is higher for GSOM with linguistic vectorization as well, 0.88 against 0.85 for linguistic vectorization and 0.87 for N-grams vectorization of cosine similarity based clustering method.

GSOM affects the FN score in the sense that the number of log entries pairs that should not be grouped into the same cluster are not actually grouped. But some log entries were misclustered into the same cluster, and hence the FP score increased. And because of this the precision score decreased.
7.2. Implemented clustering approaches

7.2.4 GSOM clustering with N-grams vectorization

The last method implemented in this master’s thesis is GSOM clustering with N-grams vectorization. Training process was performed on the same training dataset consisting of 36,257 log entries.

The training was run with 200 iterations for the training phase and 50 smoothing iterations as well. The network reached saturation after 118 iterations. The quantization error stopped decreasing significantly after 120 iterations. The resulting neural network consists of 1743 neurons, its minimal DB index is equal to 0.63, and the network can distinguish 36 clusters. The graph of the change of the quantization error during training of the GSOM network is shown in Figure 7.8. The trained GSOM is depicted in Figure 7.9. Different colors correspond to different clusters.

![Quantization Error Graph](image)

Figure 7.8: GSOM with N-grams vectorization quantization error graph

After applying the trained GSOM on the evaluation dataset, 13 clusters of log entries have been found by this approach. The diagram of log entries distribution among clusters is shown in Figure 7.10.

Evaluation metrics are collected in the Table 7.4.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total clusters found</td>
<td>13</td>
</tr>
<tr>
<td>Precision</td>
<td>0.87</td>
</tr>
<tr>
<td>Recall</td>
<td>0.92</td>
</tr>
<tr>
<td>ARI</td>
<td>0.86</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 7.4: GSOM clustering with N-grams vectorization evaluation

In comparison with GSOM clustering with linguistic vectorization precision score has increased and now it is 0.87 against 0.83, recall has decreased
7. Evaluation and Experiments

Figure 7.9: Trained GSOM with linguistic vectorization

Figure 7.10: GSOM approach clustering

(0.92 against 0.95). ARI and F-measure scores are now 0.86 and 0.90 respectively, while in GSOM with linguistic vectorization approach the corresponding values were 0.84 and 0.88.

7.2.5 Evaluation summary

The evaluation results of all four methods are summarized in the Table 7.5. Two clustering methods with Linguistic and N-grams vectorization approaches have been implemented, examined and evaluated. The first clustering methods is cosine similarity based.
### 7.2. Implemented clustering approaches

Table 7.5: Evaluation results of all considered methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Clusters</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cosine + Linguistic</td>
<td>20</td>
<td>0.97</td>
<td>0.76</td>
<td>0.85</td>
</tr>
<tr>
<td>Cosine + N-grams</td>
<td>19</td>
<td>0.98</td>
<td>0.78</td>
<td>0.87</td>
</tr>
<tr>
<td>GSOM + Linguistic</td>
<td>11</td>
<td>0.83</td>
<td>0.95</td>
<td>0.88</td>
</tr>
<tr>
<td>GSOM + N-grams</td>
<td>13</td>
<td>0.87</td>
<td>0.92</td>
<td>0.90</td>
</tr>
</tbody>
</table>

From the obtained results it can be seen that N-grams vectorization technique for cosine similarity based clustering method has an advantage over the linguistic vectorization. This is explained by the machine-generated nature of log entries. Nevertheless, despite the cosine similarity based clustering method has high precision, recall is relatively low. Log entries are partitioned into more clusters with this method since the similarity threshold is required to be high.

Another clustering method is based on GSOM neural network. For this method both the linguistic and N-grams vectorization techniques are considered as well. According to the evaluation results GSOM based clustering for both vectorization techniques has significantly better recall score since this method produces less number of cluster in comparison with cosine similarity based methods. However, the precision score for GSOM is less but these values are still permissible. Log entries are partitioned into less number of clusters, but these clusters are more generalized. Moreover, comparing GSOM with linguistic vectorization technique with GSOM with N-grams technique it can be seen that N-grams vectorization technique has an advantage over the linguistic vectorization as well.

Both clustering methods have their advantages and disadvantages. Cosine similarity based clustering provides high precision and is simpler for implementation. Moreover, this method can be used with streaming logs that is very important for business since it can help to react to possible incidents in real-time. But cosine similarity based clustering produces high number of clusters containing low number of entries. Thus the dimensionality of original log data is not reduced well.

On the other hand GSOM method has better generalization abilities and provides good dimensionality reduction of log data. The precision of partitioning is lower in comparison with cosine similarity based clustering, but is acceptable. Since the GSOM is neural network, before applying it has to be trained on dataset providing as best as possible covering of target data. Creating of such a training dataset requires good domain knowledge and specific skills. The training process itself time and computationally consumable. If the domain changes (the emergence of new types of log entries that were not included in the training dataset), then the neural network will require
additional training. All these points can be a limitation in some cases.

7.3 Correlation

Cluster analysis of logs allows to reduce their dimensionality and simplify the analysis of the obtained information. In this master’s thesis it has been assumed that it is possible to determine the root causes of anomalous server behavior using cluster analysis of log data. In the Figure 7.11 graphs of successful invocations, failed invocations and average processing time of the server for several days are depicted.

![Figure 7.11: Server load graph](image)

The peak of failed invocations is 26.02.2021. The average processing time at this date reaches 3500 ms, and failed invocations reaches more than 3000 calls.

It seems interesting to analyze what was happening on the server side in the specified period of time. An analysis of log entries clusters for the same time period has been conducted for this purpose.

Initially, the log entries of this server were clustered. Then, among all the obtained error clusters, those that correspond to the considered time period were selected. The distribution of log entries among the clusters is depicted in Figure 7.12.

From this figure it can be seen that clusters with the highest number of log entries are "Cluster 14", "Cluster 141", "Cluster 142", "Cluster 145", and "Cluster 146". Other clusters are significantly smaller, but they still contain relatively high number of log entries. From such clusters, those that contain more than 1000 log entries were selected for the detailed study. (Figure 7.13).

The log entries that make up these clusters were extracted and presented as a time series with a scale of one minute (Figure 7.14).
7.3. Correlation

Observing the graph, it can be seen that the largest number of log messages are generated in the time interval from 5 to 10 am. Moreover, the peak of generation of new error log messages falls on 6-7 am. Comparing this log entries time series with graphs on Figure 7.11 it can be noticed that peak of generation of new error log messages is correlated with the peak of failed invocations graph (Figure 7.15).
7. Evaluation and Experiments

The found correlation between the graphs of failed invocations and error clusters confirms that the analysis of clusters in different time periods allows to determine the root causes of anomalous server behavior. In the Figure 7.16 is depicted the distribution of error clusters in time. It can be seen that maximal number of errors were partitioned into clusters in the same time interval.

It seems interesting to find out the nature of errors belonging to the given clusters. The analysis of the obtained clusters revealed the errors that are
7.3. Correlation

Sown in Listings 7.1, 7.2, 7.3, 7.4, 7.5, 7.6.

```
Properties after BaseActionHander.requestMessage
fabricenterpriseId
tohttpcomplatforminstanceReferenceInstanceBeanImplcfetrackingcompositeInstanceTitleBillingNeilFarmerMESHMETRICSnulptrackingparentReferenceIdmediatorDEBBFEEBDBBADEREBBFEEBDBAreq
trackingcompositeInstanceIdOrgtrackingcompositeInstanceId
trackingecidjgDEkrPBTvFkZePGANHXeA
fabricwireSourceQueryUnbilledUsageSiebelCommsReqABCSImpl
trackingcompositeInstanceCreatedTimeFriFebUTC
trackingconversationIdnulllocalConversationIdurnDEEBBFEEBDBAtackingparentComponentInstanceIdreference
transporthttpremoteAddress
```

Listing 7.1: Cluster 141

```
Caught Exception calling an opcode
Error calling opcode
```

Listing 7.2: Cluster 1616

```
Payload after BaseActionHander.requestMessage
QueryBalanceSummaryABMoraclexmlparservXMLElementfcaaa
```

Listing 7.3: Cluster 176

```
Headers after BaseActionHander.requestMessage
```

Listing 7.4: Cluster 84

```
EBufException exception msg execOpcode ERRVALIDATIONFAILED
```

Listing 7.5: Cluster 145

```
Unable to dispatch request to httpcomcommonBusinessFaultException at oraclefabricCubeServiceEngine.requestCubeServiceEngine.java
```

Listing 7.6: Cluster 648

---

65
In this master’s thesis, a study of approaches to the analysis of server logs was carried out. One of the most important analysis methods is cluster analysis of logs. Various converting text to mathematical equivalent approaches and cluster analysis of data have been explored.

From the studied methods, two approaches to transforming text into vectors were chosen - the linguistic method and the N-gram method. The linguistic vectorization method is based on the principles of NLP and preserves the semantic structure of the analyzed text. In contrast, the N-gram method treats the server log messages as machine-generated text, without taking into account semantics.

Grouping log entries using cosine similarity and using the Growing Self-Organizing Map neural network were chosen as methods for cluster analysis of log data. Each of these methods uses both vectorization techniques. Thus, four analysis methods were considered:

- Cosine similarity based clustering with linguistic vectorization,
- Cosine similarity based clustering with N-gram vectorization,
- Growing Self-organizing neural network based clustering with linguistic vectorization,
- Growing Self-organizing neural network based clustering with n-gram vectorization.

An application implementing the selected techniques of vectorization and cluster analysis has been developed. The application was applied on a real server to analyze the log files produced by the server. A evaluation dataset was formed from the real log data, on which the selected approaches to cluster analysis of server logs were evaluated. The evaluation dataset was manually partitioned into 12 clusters. The evaluation concluded that cosine similarity clustering partitions data into more clusters containing log entries that are
Conclusion

more similar to each other. This provides cosine similarity based clustering with higher precision, but less recall. On the other hand, the neural network partitions the evaluation dataset into the number of clusters closer to the target, which provides a higher recall, but the precision of the neural network approach is lower in comparison with cosine similarity based clustering. In both approaches to log cluster analysis, better results were obtained using N-gram vectorization since the machine nature of the log messages.

The developed application was used to cluster server logs and find the correlation between the load of the real server and clusters of error log entries. The found correlation confirms that the analysis of the server logs cluster at different time periods allows to determine the root causes of the anomalous server behavior.
Future works

The neural network developed in this master’s thesis shows good results of cluster analysis of server logs. However, it has a number of limitations that complicate its use. Such limitations are mainly due to the need for preliminary training of the neural network. Forming a training sample requires advanced knowledge of the server infrastructure and the logs it generates. In addition, the use of the trained neural network in the logs generated by the server in real time is also limited, since if a new type of error occurs, which was not present in the training sample, it will lead to incorrect clustering. This limitation can be overcome by additional training of the network during its application on the log data stream. In addition, the use of other vectorization techniques has the potential to improve the clustering accuracy.

I want to devote my PhD research to the development of algorithms for additional training of a neural networks for stream processing of log data and using of various methods of vectorization.
Bibliography


Bibliography


Acronyms

CLI  Command Line Interface
CSV  Comma-Separated Values
NLP  Natural Language Processing
SaaS System as a Service
LSTM Long Short-Term Memory
BLSTM Bidirectional Long Short-Term Memory
PC  Personal Computer
ELK  Elasticsearch, Logstash, Kibana
JSON JavaScript Object Notation
GUI  Graphical User Interface
XLS  Excel Spreadsheet
PDF  Portable Document Format
HTML HyperText Markup Language
SOAP Simple Object Access Protocol
REST Representation State Transfer
API  Application Programming Interface
HTTP HyperText Transfer Protocol
TLS  Transport Layer Security
SOA  Service Oriented Architecture
A. Acronyms

**ESB**  Enterprise Service Bus
**J2EE**  Java Enterprise Edition
**EJB**  Enterprise Java Beans
**RMI**  Remote Method Invocation
**JMS**  Java Messaging Service
**NER**  Named Entity Recognition
**NLTK**  Natural Processing Toolkit
**TF**  Term Frequency
**TF-IDF**  Term Frequency – Inverse Document Frequency
**ANN**  Artificial Neural Network
**MLP**  Multi-Layered Perceptron
**SOM**  Self-Organizing Map
**GSOM**  Growing Self-Organizing Map
**BMU**  Best Matching Unit
**GT**  Growth Threshold
**TE**  Total Error
**ARI**  Adjusted Rand Index
Appendix B

Contents of enclosed CD

readme.txt ....................... the file with CD contents description
src.................................. the directory of source codes
  application.......................... implementation sources
    analyzer................ the directory of source codes of the application
    cluster_analysis, the directory containing server reports and the server load figure
    clusters........... the directory where the clusters can be stored
    imgs .... the directory containing images generated during analysis
    reports.. the directory containing the generated clustering reports
    trained .. the directory containing trained GSOM neural networks
  Correlation_analysis.ipynb ...... the file containing correlation analysis
  Evaluation.ipynb ... the file containing the evaluation analysis of implemented clustering methods
  evaluation_dataset.out.... the file containing evaluation dataset
  evaluation_dataset.out... the file containing manually clustered and labeled evaluation dataset
  requirements.txt .. the file containing the list of required libraries
  thesis............. the directory of LATEX source codes of the thesis
  text................................. the thesis text directory
  thesis.pdf........................... the thesis text in PDF format
Class diagram is a graphical representation of the application in the object oriented way. Such a diagram represents interactions among the individual entities of the domain, and also describes their inner structure and types of relationships. Class diagram of the application developed in this master’s thesis is shown in Figure C.1.
Figure C.1: Class diagram of the developed application
This chapter contains a manual describing how to install the application developed in this master’s thesis and the guide how to use it.

The application was used and tested on OS Linux with installed Python 3.8.

D.1 Installation manual

D.1.1 Copy the application

Copy the application source codes from the folder src/application on the CD to your computer.

D.1.2 Install required packages

Go to the application folder on your computer and create virtual environment via the following command

```
python3 -m venv __venv__
```

After successful creating of the virtual environment activate it via:

```
__venv__/bin/activate
```

install all required packages via

```
pip3 install -r requirements.txt
```
D. INSTALLATION MANUAL AND USER GUIDE

D.2 Usage guide

D.2.1 Run the application

Once all of the required packages are installed, run the application via the following command in command line:

```
python analyzer/main.py --help
```

In the console should be the following output:

```
$ python analyzer/main.py --help
Usage: main.py [OPTIONS] COMMAND [ARGS]...

Log Analysis Tool. Parses and clusters log files. Reports are saved to 'reports' folder.

Options:
  --version    Show the version and exit.
  --help       Show this message and exit.

Commands:
  cos  Runs clustering based on cosine similarity.
  gsom Runs GSOM clustering.
  method-eval Runs evaluation of the chosen model.
```

Listing D.1: CLI for helping info about the developed application for log clustering

D.2.2 Cosine similarity based clustering

To cluster input log data with cosine similarity based method use the `cos` as it was described in Listing D.1. To get a helping info about the parameters for cosine similarity based clustering run following command:

```
python analyzer/main.py cos --help
```

The helping info about cosine similarity based clustering should be:

```
$ python analyzer/main.py cos --help
Usage: main.py cos [OPTIONS] SOURCE...

Runs clustering based on cosine similarity.

Options:
  -V, --vectorization STRING Vectorization type (ngrams or tf-idf).
    [default: ngrams]
  -n NUMBER    Ngrams dimensionality. [default: 2]
  -t, --threshold REAL Similarity threshold value. [default: 0.995]
  -S, --save    Save results of clustering to 'clusters'
  --help       Show this message and exit.
```

Listing D.2: CLI for helping info about Cosine similarity based clustering

To perform cosine similarity based clustering with N-grams vectorization run the following command:
The application performs cosine similarity based clustering with N-grams vectorization, generates and saves the clustering report, and saves the obtained clusters. The result of clustering should be like it is presented in the following Listing:

```
$ python analyzer/main.py cos -S ./evaluation.out
Starting...
Loading file: ./evaluation.out
Log entries count: 120
Clustering started...
Total clusters found: 19
Clustering successfully done. Time elapsed: 0.3805074691772461s
Clustering report ngrams_cos_clustering_report_evaluation.csv saved to 'reports' folder.
Saving clusters...
Clusters saved into 'clusters' folder.
```

Cosine similarity based clustering with Linguistic vectorization can be performed in the similar way:

```
$ python analyzer/main.py cos -S -V tf-idf ./evaluation.out
Starting...
Loading file: ./evaluation.out
Log entries count: 120
Clustering started...
Total clusters found: 20
Clustering successfully done. Time elapsed: 1.816401720046997s
Clustering report tf-idf_cos_clustering_report_evaluation.csv saved to 'reports' folder.
Saving clusters...
Clusters saved into 'clusters' folder.
```

D.2.3 GSOM based clustering

get a helping info about the GSOM based clustering run following command:

```
python analyzer/main.py gsom --help
```

The helping info about cosine similarity based clustering should be:

```
$ python analyzer/main.py gsom --help
Usage: main.py gsom [OPTIONS] SOURCE...
```
D. INSTALLATION MANUAL AND USER GUIDE

Runs GSOM clustering. 

Or path(s) to preparse log entries .pickle file(s).

Options:

- `--spread-factor REAL` Spread factor. [default: 0.83]
- `--smooth-factor REAL` Smoothing factor. [default: 0.8]
- `--alpha REAL` Alpha. [default: 0.9]
- `--learning-rate REAL` Learning rate. [default: 0.3]
- `--distribution REAL` Factor of distribution. [default: 0.3]
- `--neighbourhood NUMBER` Max neighbourhood around the winner neuron.[default: 6]
- `--R REAL` R parameter. [default: 3.8]
- `--gsom PATH` Path to existing trained GSOM.
- `--training PATH` Path to training data.
- `--training-epochs NUMBER` Total numbers of epochs for training. [default: 150]
- `--smoothing-epochs NUMBER` Total numbers of epochs for smoothing. [default: 150]
- `--save` Save results of clustering to 'clusters' folder.
- `--save-model STRING` Save trained GSOM model to a file with given name. Timestamp will be added to the name. Example: gsom_<timestamp>.pickle [default: gsom]
- `--vectorization STRING` Vectorization type (ngrams or tf-idf). [default: ngrams]
- `--vectors_file PATH` Path to vectors file. It is required for TF-IDF vectorization.
- `--n NUMBER` Ngrams dimensionality. [default: 1]
- `--help` Show this message and exit.

Listing D.3: CLI for helping info about GSOM based clustering

Every parameter of GSOM model can be adjusted by user. Nevertheless, there are default values for each parameter.

Training of new GSOM model looks like:

```
$ python analyzer/main.py gsom -t ./evaluation.out ./evaluation.out
Starting...
Loading training data from file: ./evaluation.out
Training data set size: 170
Loading file: ./evaluation.out
Log entries count: 120
Training started. Press CTRL + C to stop training phase and proceed to smoothing.
Min DB index value is 0.4934572511842771 for 2 clusters. Current neurons count: 11, Saturation: 7, QE: 0.22690244545454544
Min DB index value is 0.09939134017016994 for 4 clusters. Current neurons count: 23, Saturation: 12, QE: 0.23877245652173915
Min DB index value is 0.26937708805085536 for 2 clusters. Current neurons count: 29, Saturation: 6, QE: 0.30479293103448285
```
D.3 Analysis of clusters

The resulting clustering reports are used for analysis purposes. The analysis was performed in Jupyter Lab.