TECHNOLOGIES AND ENGINEERING

Journal homepage: https://technologies-engineering.com.ua/en
Vol. 26, No. 2, 2025

Received: 06.12.2024
Revised: 18.03.2025
Accepted: 30.04.2025

UDC 004.62

DOI: 10.30857/2786-5371.2025.2.2

Impact of log file processing on learning speed and defect classification accuracy

Anton Kaiafiuk[*]

Postgraduate Student
Kyiv National University of Technologies and Design
01011, 2 Mala Shyianovska Str., Kyiv, Ukraine
https://orcid.org/0009-0003-9917-0834

Abstract. The purpose of the study was to investigate the effect of automatic testing log file preprocessing on the speed of vectorisation and training of machine learning models. The HDFS_v3_TraceBench set was used, which contains more than 370 thousand traces collected in the Hadoop Distributed File System Environment. Processing included noise removal, lemmatisation, and duplication reduction. The data was vectorised using the Term frequency – inverse document frequency method, and then the RandomForestClassifier model was trained. The experimental results showed that optimising the input data reduced the total processing time by almost five times. The time required for text vectorisation and model training has been reduced, which helped to speed up work with large volumes of logs. However, the classification accuracy was not only preserved, but also showed a slight improvement: the F1-score and Matthews correlation coefficient indicators remained consistently high. There was also a decrease in the Log Loss value, which indicated an increase in the model's confidence in its own forecasts. This is especially important in the context of unbalanced classes that are characteristic of defect classification problems. A detailed analysis showed that a significant part of the service and repetitive information in the logs is not critical for training the model, and its removal, on the contrary, improves the quality of data preparation. In the course of the study, it was also confirmed that the resulting target labels for logs correspond to typical error classes. Implemented log file processing not only reduces computational costs, but also supports or improves the quality of forecasting. These results confirmed the feasibility of including the log cleaning and optimisation step in the overall process of building machine learning models for automated testing. The results obtained can be integrated into automated pipelines for classifying defects and generating bug reports. This will help to reduce the amount of manual labour and increase the efficiency of teams

Keywords: regular expressions; lemmatisation; vectorisation; machine learning; test automation

Introduction

Automated testing of the latest software generates large volumes of unstructured log files, which are complicated by the growing number of tests and the complexity of systems. According to R. Peronto (2024), the annual growth in the volume of log files is 250%. Traditional log analysis methods are expensive and slow, which leads to delays in detecting defects and releasing software. Machine learning allows automating this process, but the effectiveness of models largely depends on the quality of preprocessing: redundant information slows down learning, and excessive filtering reduces accuracy. The balance between these extremes determines the performance of the system, especially in problems of classification of defect types, where it is important not only to identify the problem, but also to understand its nature.

Recent international studies have mainly focused on detecting anomalies in logs. R.R. Abdalla & A.K. Jumaa (2022) in their study showed the effectiveness of ML (Machine Learning) when working with unstructured journals, but the classification of known defects was not considered. S. Ramachandran *et al.* (2023) proposed a model based on deep neural networks and custom "LogWord2Vec" vectorisation for classifying errors in large log files, achieving high accuracy, although the impact of preprocessing on resource consumption was ignored. Log vectorisation is a key step in preparing for analysis. TF-IDF (Term

[*]Corresponding author

frequency – inverse document frequency) remains one of the most common approaches for highlighting informative words. A. Sandhu & S. Mohammed (2022) noted that frequently repeated terms reduce the quality of features, so limiting the number of duplicates helps reduce their impact while maintaining the semantic load of messages. In application systems, log analytics is increasingly used to detect states or types of errors. A. Brandão & P. Georgieva (2020) was one of the first to apply ML to network logs to detect attacks. P. Ryciak *et al.* (2022) adapted this approach to search for anomalies in system logs using sequential models. E. Shirzad & H. Saadatfar (2022) showed that analysis of patterns in unstructured logs allows predicting failures in a Hadoop cluster, which increases the reliability of the system. Y. Huangfu (2022) investigated the use of various machine learning approaches to diagnose log-based software failures, emphasising the importance of data preprocessing to improve model efficiency. Q. Qin *et al.* (2024) proposed a two-step approach to log file processing, using semi-supervised learning to speed up log classification in large online systems. Z.A. Khan *et al.* (2024) conducted an empirical study of the effect of log parsing on the accuracy of anomaly detection, emphasising that the distinguishing property in parsing results is key to achieving high accuracy. The problem of preprocessing log files was also actively investigated by Ukrainian researchers. In particular, M. Prodeus *et al.* (2024) proved that standardisation, normalisation, and feature selection significantly improve Random Forest results in network data anomaly detection problems. Despite the differences in domains, this confirmed the importance of filtering unnecessary technical information in logs to improve the efficiency of models. S. Kapitanets & G. Radelchuk (2022) emphasised the importance of orderly data logging for timely detection of failures in software systems, noting that chaotic logging practices significantly complicate error detection. O. Khil & V. Yakovina (2023) emphasised in their study that the success of the model largely depends on the quality of input data. The study by S.A. Hussein & S.R. Répás (2024) provided an overview of modern methods for detecting anomalies in log files using machine learning algorithms. The researchers analysed the effectiveness of statistical, ML, and DL approaches in the context of variable structure and large amounts of log data, highlighting the growing role of AI in strengthening cybersecurity.

The purpose of this study was to investigate the influence of log file preprocessing methods on the efficiency of machine learning in automatic defect classification. In particular, it was considered how noise removal using regular expressions, lemmatisation, and TF-IDF vectorisation affects the speed of converting text data to numeric vectors and model training.

Materials and Methods

To achieve this goal, a set of methods of scientific cognition was applied, including empirical and experimental approaches. In particular, a computational experiment was conducted: a data processing and analysis pipeline was implemented, on which various scenarios for preprocessing log files were tested. The methods of cluster analysis (for preliminary detection of structures in data) and computer experiment were used to evaluate the performance of text data preprocessing, the learning time of the machine learning model, and the accuracy of model prediction. The results of various data preparation methods were compared with each other (comparative analysis method) to identify their impact on speed and accuracy. This approach allows drawing reasonable conclusions about the causal relationships between the applied data transformations and the obtained ML model metrics.

The open dataset HDFS_v3 (TraceBench), a collection of log files obtained in the Hadoop distributed file system (HDFS) during cluster operation (Zhou *et al.*, 2014). This set contains more than 370,000 trace records (log messages) from the real infrastructure-as-a-service (IaaS) environment. Below is the programme code for calculating the silhouette score based on a data set:

```
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.mixture import GaussianMixture

X = X.toarray()
range_n_clusters = [3, 4, 5, 6]

fig, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = axes.flatten()

for idx, n_clusters in enumerate(range_n_clusters):
    gmm = GaussianMixture(n_components = n_clusters, random_
state = 42)
    cluster_labels = gmm.fit_predict(X)

    # Calculate avg silhouette coefficient
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(
        f"n_clusters = {n_clusters}, average silhouette
coefficient = {silhouette_avg:.4f}"
    )

    # Calculate silhouette coefficient for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    ax = axes[idx]
    ax.set_title(
        f"Number of clusters: {n_clusters}\nAverage silhouette
coefficient: {silhouette_avg:.4f}"
    )
    ax.set_xlabel("Silhouette coefficient")
    ax.set_ylabel("Cluster")

    ax.set_xlim([-0.1, 1])
    ax.set_ylim([0, len(X) + (n_clusters + 1) * 10])
    ax.axvline(x = silhouette_avg, color = "red", linestyle = "--")

    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_
values[cluster_labels = = i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor = color,
            edgecolor = color,
            alpha = 0.7,
        )

        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10  # 10 for space between clusters

    ax.set_yticks([])  # Hide sticks for Y axis
    ax.set_xticks(np.arange(-0.1, 1.1, 0.2))

plt.tight_layout()
plt.show()
```

Since the purpose of the study was to train the ML model to classify defect types, and the HDFS_v3_TraceBench dataset contains only log files without corresponding labels, the first step was to create target classes (labels) for each log file. The first step was to read the log files and create a pandas.DataFrame structure with error messages; next, the data were vectorised using the TF-IDF method and clustered using Gaussian mixture; the optimal number of clusters was determined using the silhouette score, according to A. Géron (2022), and creating a pandas.DataFrame with target classes for further model training. Ultimately, the obtained clusters were analysed and compared with the types of defects given in the official documentation according to the HDFS architecture guide (n.d.).

The next step was data preprocessing. Problems identified in logs: unstructured, redundant information and noise. To solve these problems, the following pretreatment steps were implemented (Fig. 1): using regular expressions, timestamps, logging levels (INFO, DEBUG, ERROR, WARN), ID (Identification Number), IP addresses, UUID (Universal Unique Identifier) and extra special characters were removed; words were returned to their original form to improve the quality of vectorisation and data consistency; the number of duplicate messages was reduced to reduce the amount of data.

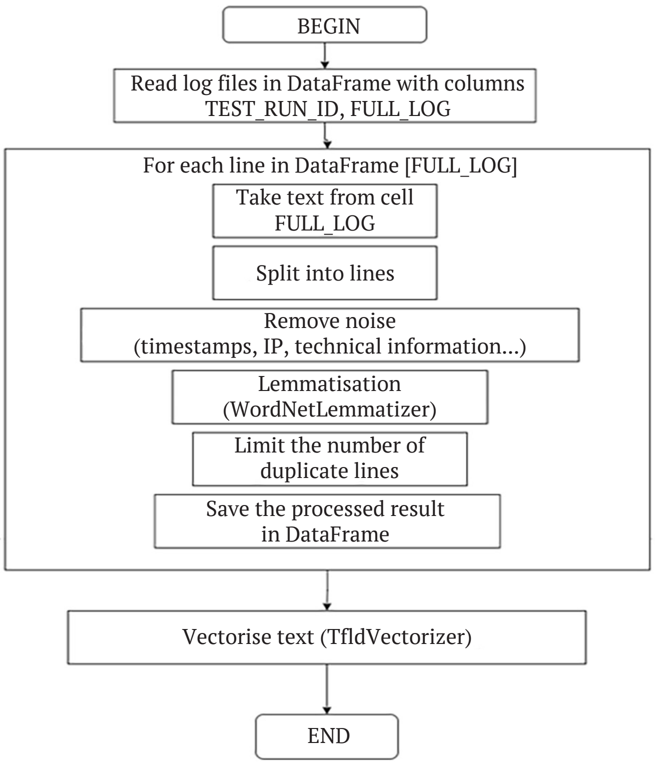

BEGIN
Read log files in DataFrame with columns TEST_RUN_ID, FULL_LOG
For each line in DataFrame [FULL_LOG]
Take text from cell FULL_LOG
Split into lines
Remove noise (timestamps, IP, technical information...)
Lemmatisation (WordNetLemmatizer)
Limit the number of duplicate lines
Save the processed result in DataFrame
Vectorise text (TfldVectorizer)
END

***Figure 1**. Block diagram of log file preprocessing*
***Source:** developed by the author based on research*

After preprocessing the data, the TF-IDF (Term frequency – inverse document frequency) method was used to vectorise the text, implemented using the TfidfVectorizer library from the scikit-learn package, according to the TfidfVectorizer (n.d.). This method was widely used for processing text data, as it allows determining the weight of each word based on its frequency in the document and rarity throughout the corpus, according to G. Salton *et al*. (1975). TF-IDF helps to reduce the impact of noisy or overly frequent words by highlighting key terms that are unique or relevant to specific messages or errors.

Equation for calculating TF:

$$\mathrm{TF}(t,d) = \frac{\text{Number of times the term } t \text{ appears in the document } d}{\text{Total number of terms in the document } d}, \quad (1)$$

where TF – frequency of the term, t – the term being analysed, d – the document for which the analysis is being performed.

Equation for calculating IDF:

$$\mathrm{IDF}(t,D) = \frac{\text{Total number of documents in the corpus } D}{\text{Number of documents including the term } t}, \quad (2)$$

where IDF – inverse frequency of documents, t – the term being analysed, D – entire body of documents.

Equation for calculating TF-IDF:

$$\mathrm{TF\text{-}IDF}(t, d, D) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t, D), \quad (3)$$

where TF-IDF – combined metric that considers the frequency of the term in the document and its rarity throughout the body, t – the term being analysed, d – the document for which the analysis is performed, D – entire body of documents, TF – term frequency, IDF – inverted document frequency.

To implement this pipeline, the Pipeline interface was used, which inherits the BaseEstimator classes (BaseEstimator section in scikit-learn official documentation, n.d.) and TransformerMixin (TransformerMixin, n.d.), according to Developing scikit-learn estimators (n.d.):

```
import re
from typing import Union, Optional
from sklearn.base import BaseEstimator, TransformerMixin
from nltk.stem import WordNetLemmatizer


class RegexpCleaner(BaseEstimator, TransformerMixin):
  “””Removes substrings for the given patterns.”””

  def __init__(self, patterns: Union[str, list[str]]) -> None:
    “””Initialize with a single pattern, or list of patterns to
remove.

    :param patterns: e.g. r’[^a-zA-Z\s]’, [r’[^a-zA-Z\s]’, ...]
    “””
    self.patterns: list[str] = [patterns] if isinstance(patterns, str)
else patterns

  def fit(self, X: list[str], y:
Optional[list] = None) -> “RegexpCleaner”:
    “””Dummy function to follow the interface of a
transformer.”””
    return self

  def transform(self, X: list[str], y: Optional[list] = None) ->
list[str]:
    “””For each line in X remove given pattern one by one.

    :param X: dataset to transform
    :param y: added to implement interface.
    :return: transformed dataset
    “””
    cleaned: list[str] = []
    for text in X:
      for pattern in self.patterns:
        text = re.sub(pattern, “”, text)
      cleaned.append(text)
    return cleaned

class Lemmatizer(BaseEstimator, TransformerMixin):
  “””Lemmatizes data in a given dataset”””

  def __init__(self) -> None:
    self.lemmatizer = WordNetLemmatizer()

  def fit(self, X: list[str], y: Optional[list] = None) -> “Lemmatizer”:
    “””Dummy function to follow the interface of a
transformer.”””
    return self

  def transform(self, X: list[str], y:
Optional[list] = None) -> list[str]:
    “””
    Lemmatize the lines in the given dataset.
    “””
    lemmatized: list[str] = []
    for text in X:
      words = text.split()
      lemmatized_words = [self.lemmatizer.lemmatize(word) for
word in words]
      lemmatized.append(“ “.join(lemmatized_words))
    return lemmatized


class DuplicateLimiter(BaseEstimator, TransformerMixin):
  “””Limits number of duplicates in a given dataset”””

  def __init__(self, max_dupes: int = 5) -> None:
    “””Initialize with a number of duplicates to leave

    :param max_dupes: number of duplicates to leave
    :raises ValueError: if max_dupes has an incorrect type
    “””
    if not isinstance(max_dupes, int):
      raise ValueError(f”max_dupes should be integer,
{type(max_dupes)} was given.”)
    self.max_dupes = max_dupes

  def fit(self, X: list[str], y:
Optional[list] = None) -> “DuplicateLimiter”:
    “””Dummy function to follow the interface of a
transformer.”””
    return self

  def transform(self, X: list[str], y: Optional[list] = None) -> list[str]:
    “””Limit the duplicates in a dataset

    :param X: dataset to transform
    :param y: added to implement interface.
    :return: transformed dataset
    “””
    line_counts = {}
    result = []
    for line in X:
      line_counts[line] = line_counts.get(line, 0) + 1
      if line_counts[line] < = self.max_dupes:
        result.append(line)
    return result
```

Such a pipeline for data preprocessing has a concise structure and can be easily expanded if necessary:

```
from sklearn.pipeline import Pipeline

ID_PATTERN = r’\b[A-F0-9]{16},’
SPECIAL_CHAR_PATTERN = r”[^a-zA-Z\s]”
MULTIPLE_WHITESPACES_PATTERN = r”\s+”

pipeline = Pipeline([
  (‘regex_cleaner’, RegexpCleaner(patterns=[ID_PATTERN,
SPECIAL_CHAR_PATTERN, MULTIPLE_WHITESPACES_
PATTERN])),
    ('lemmatizer', Lemmatizer()),
    ('dup_limiter', DuplicateLimiter(max_dupes = 5)),
])

X_preprocessed = pipeline.fit_transform(X)
```

The last step was to select the model and metrics for evaluation. The RandomForestClassifier algorithm, which is an ensemble machine learning method known for its high accuracy and resistance to retraining in classification problems, was chosen to classify defect types. Previous studies confirm its effectiveness under similar conditions, according to M. Prodeus *et al.* (2024). Three metrics were used to evaluate the model quality: Log Loss, F1-score, and MCC (Table 1). Each of them measures different aspects of the algorithm's performance, which is especially important for working with unbalanced classes that are often found when classifying defects in log files, according to C. Cao *et al.* (2020).

***Table 1**. Metrics for evaluating the model*

Metric	What it measures	Why it is used
Log Loss	How confident is the model in its predictions	Considers the probability of predictions
F1-score	Balance precision and recall	Important for unbalanced classes
MCC	General correlation of predictions with real classes	Best metric for unbalanced data

***Source:** developed by the author based on C. Cao et al. (2020)*

The use of these metrics allows getting a comprehensive understanding of the quality of the classification model, ensuring the reliability and efficiency of automated analysis of defects in log files.

In this context, the programme code that was used to create and train the model is presented:

```
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_
size = 0.7, random_state = 42, stratify = y)

rf_clf = RandomForestClassifier(random_state = 42, n_
estimators = 100)
start_time_train = time.time()
rf_clf.fit(X_train, y_train)
train_time = time.time() - start_time_train

y_pred = rf_clf.predict(X_test)
predict_proba = rf_clf.predict_proba(X_test)

log_loss_ = log_loss(y_test, predict_proba)
f1 = f1_score(y_test, y_pred, average="weighted")
matthews = matthews_corrcoef(y_test, y_pred)
print(f"log_loss: {log_loss_:.4f}, f1: {f1}, matthews: {matthews},
train_time: {train_time}")
```

Results and Discussion

This section presents the results of an experimental study and their analysis. First, the obtained cluster structures and the formation of defect classes were considered, followed by the effect of pre-processing on the learning rate and model accuracy. In the end, a comparison is made with the latest developments of other researchers in the field of log file analysis. This approach ensured the integrity of the presentation and allowed interpreting the results qualitatively.

The obtained cluster analysis data confirmed the presence of five groups (clusters) in the data sample. The values of the silhouette coefficient for a different number of clusters were: for 3 clusters – 0.3758 (unsatisfactory separation); 4 – 0.4770 (improvement, but still weak); 5 – 0.5471 (optimal balance of cohesion and separation); 6 – 0.6088 (higher result, but there is an appearance of overtraining and blurring of borders between some clusters). Thus, the choice of 5 clusters as the optimal option for constructing classification labels is justified. Figure 2 showed the results of the silhouette coefficient analysis for a different number of clusters, confirming the above values.

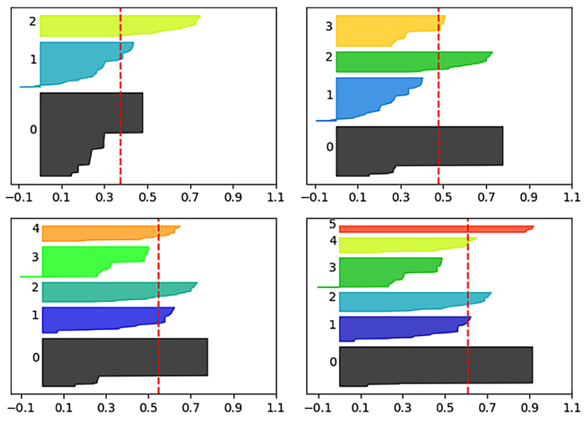


***Figure 2**. Silhouette factor analysis based on the number of clusters*
***Source:** developed by the author based on research*

Figure 3 showed the result of clustering using an uncontrolled Gaussian Mixture Model. For clarity, the data dimension was reduced to two main components using the PCA (Principal component analysis) method, which allows mapping clusters in two-dimensional space.

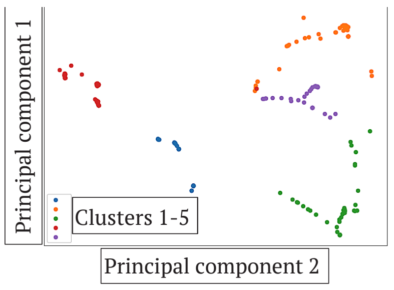


***Figure 3**. Visualisation of clusters obtained during the previous step*
***Source:** developed by the author based on research*

The resulting clusters look clearly separated, which indicates a successful choice of model parameters. The result corresponds to the previous silhouette analysis, according to which the optimal number of clusters is 5-6. In this case, the model also formed 5 clusters, which confirms the consistency between different analysis methods.

Next, a dataset was generated with labels (target values) for each log file from the dataset. Figure 4 showed the distribution of the sample by five classes of defects (codes 0-4) detected during automated testing. The X-axis shows the codes of defect classes, and the Y-axis shows the number of samples in each class. According to the HDFS architecture guide (n.d.), the resulting uneven distribution corresponds to the types of defects that were removed as part of the use of the system under study. It displays the actual types of failures, which include: data disk failure, heartbeats and re-replication, cluster rebalancing, data integrity, and metadata disk failure.

The distribution between classes was uneven, which indicates a more frequent occurrence of certain types of defects. In particular, class 0 (red) contains the largest number of samples, while classes 2 (purple) and 4 (pink) contain the least. This imbalance is typical for large distributed systems and affects further training of models, because models can lean towards the dominant class.

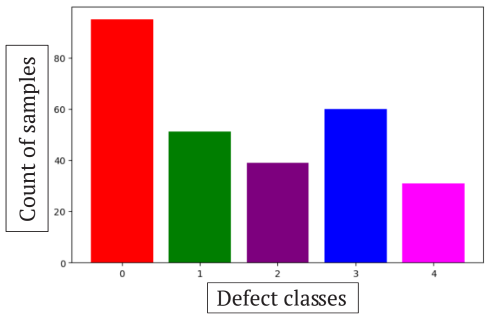
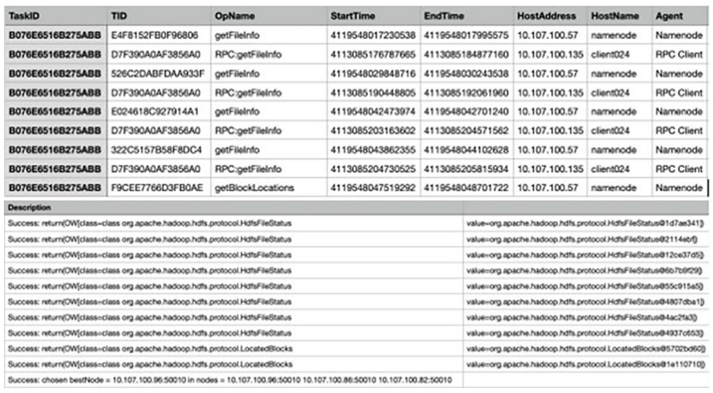


Figure 4. *Distribution of the sample by five classes of defects*
Source: *developed by the author based on research*

To illustrate the input data, Figure 5 showed a fragment of a raw log file from the HDFS_v3_TraceBench dataset. The file contains a significant amount of noise and duplicate service records. This format is typical for log files generated by software systems or automated tests, and confirms the presence of unstructured, redundant information in logs.



TaskID	TID	OpName	StartTime	EndTime	HostAddress	HostName	Agent
B076E6516B275ABB	E4F8152FB0F96806	getFileInfo	4119548017230538	4119548017995575	10.107.100.57	namenode	Namenode
B076E6516B275ABB	D7F390A0AF3856A0	RPC:getFileInfo	4113085176787665	4113085184877160	10.107.100.135	client024	RPC Client
B076E6516B275ABB	526C2DABFDAA933F	getFileInfo	4119548029848716	4119548030243538	10.107.100.57	namenode	Namenode
B076E6516B275ABB	D7F390A0AF3856A0	RPC:getFileInfo	4113085190448805	4113085192061960	10.107.100.135	client024	RPC Client
B076E6516B275ABB	E024618C927914A1	getFileInfo	4119548042473974	4119548042701240	10.107.100.57	namenode	Namenode
B076E6516B275ABB	D7F390A0AF3856A0	RPC:getFileInfo	4113085203163602	4113085204571562	10.107.100.135	client024	RPC Client
B076E6516B275ABB	322C5157B58F8DC4	getFileInfo	4119548043862355	4119548044102628	10.107.100.57	namenode	Namenode
B076E6516B275ABB	D7F390A0AF3856A0	RPC:getFileInfo	4113085204730525	4113085205815934	10.107.100.135	client024	RPC Client
B076E6516B275ABB	F9CEE7766D3FB0AE	getBlockLocations	4119548047519292	4119548048701722	10.107.100.57	namenode	Namenode

Description	
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@1d7ae341])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@2114ebf])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@12ce37d5])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@6b7b9f29])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@55c915a5])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@4807dba1])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@4ac2fa3])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.HdfsFileStatus	value=org.apache.hadoop.hdfs.protocol.HdfsFileStatus@4937c653])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.LocatedBlocks	value=org.apache.hadoop.hdfs.protocol.LocatedBlocks@5702bd60])
Success: return(OW[class=class org.apache.hadoop.hdfs.protocol.LocatedBlocks	value=org.apache.hadoop.hdfs.protocol.LocatedBlocks@1e110710])
Success: chosen bestNode = 10.107.100.96:50010 in nodes = 10.107.100.96:50010 10.107.100.86:50010 10.107.100.82:50010	

Figure 5. *Log file data format from the HDFS_v3_TraceBench dataset*
Source: *J. Zhu et al. (2013)*

The comparative analysis performed showed a clear advantage of the model trained on cleared logs over the option without preprocessing. First, the volume and structure of input data changed significantly: deleting duplicate and service records reduced the volume of log files by almost 80%, reducing the dimension of the feature space. Optimising the input data significantly reduced the training time of the model, as shown in Figure 6. The first column showed the time spent on training (red), and the second column showed the time spent on preprocessing (blue) and training. The results showed that although data preprocessing takes time, the total time spent processing data and training the model is almost five times less compared to training the model on raw data. In particular, the conversion of log files to TF-IDF vectors and further training of the model on cleaned data was much faster than on raw data. This is important from the standpoint of scaling, because it helps to speed up the analysis of large volumes of log files without additional resources. In addition, the preprocessing process can take place separately from vectorisation and training, to save resources or save disk space spent on storing logs.

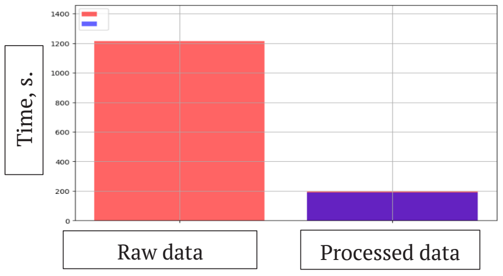


Figure 6. *Impact of data quality on time spent on vectorisation*
Source: *developed by the author based on research*

The simulation results showed that pretreatment of data contributed not only to the preservation, but also to the improvement of classification accuracy. Figure 7 compared the performance of the model trained on cleaned data with the model built on raw input data. There was a moderate increase in the values of the F1 measure and the Matthews correlation coefficient (MCC) after cleaning, which indicated a decrease in noise levels and the preservation of key informative features. A slight increase in the value of the Log Loss metric did not significantly affect the overall quality of classification, which indicates that the model remains confident in its forecasts. Thus, the model trained on cleared logs demonstrated both acceleration and high stability of accuracy metrics. Consequently, high-quality preprocessing of logs provided simultaneous acceleration of work and increased reliability of the model, which is a significant advantage from the standpoint of practical application in defect analysis.

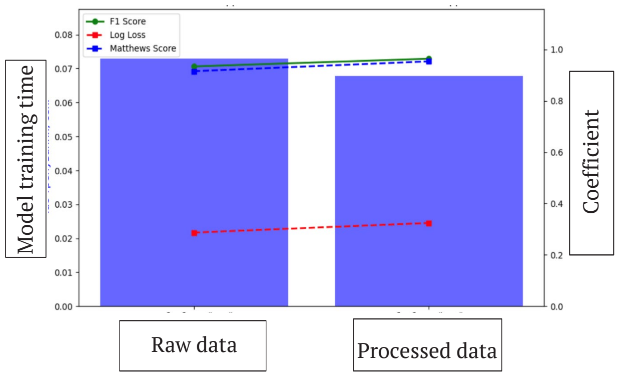


Figure 7*. Influence of data quality on model accuracy*
Source: *developed by the author based on research*

The results obtained in the framework of this study are consistent with the conclusions of modern works in the field of analysis of system journals. In particular, M. Aljabri *et al.* (2022) successfully applied Machine Learning and Deep Learning algorithms to classify network firewall records, achieving very high accuracy (more than 99% for Random Forest). This is consistent with the current study, where the Random Forest model also showed consistently high performance on cleaned data. This coincidence of results indicates the universality of the chosen approach to log classification, even if the data nature is different. Similar results were observed in the study by V. Koval *et al.* (2022), where the Random Forest Regression model was used to predict the quality of electricity in systems with renewable sources. Despite the different scope of application, the model demonstrated high stability and accuracy when working with a large array of synchronised data. This confirms the flexibility and reliability of Random Forest in tasks that require processing complex structured logs, and reinforces the argument for the universality of this approach in classification and forecasting problems. Other researchers have also emphasised the importance of effective preprocessing of journals. Thus, J. Wang *et al.* (2020) proposed a LogEvent2Vec model that directly feeds log events to the input of a word2vec network for vectorisation. This helped to avoid multi-level transformations and speed up data analysis by about 30 times, while improving the accuracy of anomaly detection. Although the TF-IDF approach presented in this paper is simpler, it also provided approximately five-fold acceleration, which confirms the general trend: optimising the log representation significantly affects the performance of models and allows adapting the analysis to limited computing resources. S.A. Hussein & S.R. Répás (2024) noted the advantages of semantic features (based on word2vec) over purely statistical features (TF-IDF) for anomaly recognition tasks in system logs. However, using the word2vec technique itself requires significant computational resources and careful dictionary construction. In the case of the current study, even the use of a relatively simple TF-IDF proved effective due to preliminary clearing of logs, which reduced noise and highlighted meaningful terms, which allowed compensating for the lack of a deeper semantic context. Moreover, the results obtained are consistent with the findings of Z.A. Khan *et al.* (2024), who showed that it is not so much the absolute accuracy of log parsing as the ability to distinguish informative characteristics ("distinctiveness") that determines the success of models. In this context, the experiment of the current study was aimed precisely at improving the selection of features through data cleaning, which ultimately allowed improving both the speed and accuracy of the model without complicating the architecture. Thus, the comparison confirmed the hypothesis that high-quality preprocessing of log files is a key factor in both speeding up calculations and improving the accuracy of defect prediction, which is consistent with trends in current research. U. Meteriz *et al.* (2020) presented a log classification system for telecom systems using a deep neural network. The researchers noted the success of advanced NLP (Natural Language Processing) for logs, which opens up prospects for automated analysis of complex technical messages. The results of the current study are consistent with their classification accuracy, although the approach used in the current study (TF-IDF + RandomForest) is significantly simpler and resource-saving compared to the deep CNN (Convolutional Neural Network) model. The effectiveness of data cleaning was also confirmed by the findings of O. Johnphill *et al.* (2024), according to which removing uninformative records and duplicate messages allows the model to focus on relevant signals, which increases learning efficiency. D.A. Bhanage & A.V. Pawar (2023) presented an approach to classifying system logs as natural language texts. The researchers tested a combination of several vectorisation techniques – TF-IDF, Word2Vec, and tonality analysis (polarity) – in combination with classical classifiers. According to the results, the most promising approach was to consider the "tone" of messages, which improved the recognition of complex irregular logs. D.A. Bhanage & A.V. Pawar's approach demonstrated that additional semantic features (such as sentiment) can improve the quality of log analysis.

The method used in this study, on the contrary, is deliberately simplified (only TF-IDF without deep semantic analysis), which makes it less resource-intensive, although it is somewhat inferior in terms of contextual nuances. P. Marjai & A. Kiss (2024) pointed out in their paper that pre-structuring logs helps the model to better generalise error messages. The results obtained in the current study confirmed this trend: cleaning and normalising text data from log files helps to improve classification accuracy even without complicating the vectorisation approach. This is especially important in application environments where resources are limited and data processing speed requirements are critical.

The results showed that the quality and volume of log data have a complex relationship with the performance of machine learning models. In contrast to the common practice of accumulating and storing complete logs, the conducted experiment indicates the possibility of significantly reducing the amount of input information without negatively affecting the accuracy of classification. In particular, the removal of service records, repetitions, and technical labels helped not only to reduce processing time, but also to achieve better classification quality by key metrics. This effect can be explained by a decrease in the influence of information noise, which usually overloads the model and reduces its ability to distinguish relevant patterns in logs. The presence of a regular structure in messages after lemmatisation and cleaning can also help improve the vectorisation process, especially when using methods such as TF-IDF. However, the results indicate the potential adaptability of the proposed approach to other logging systems.

Conclusions

In the course of the study, it was found that log file preprocessing, which includes noise removal, lematisation, and limiting message duplication, significantly improves the efficiency of building machine learning models for defect classification. In particular, it has been shown that the use of regular expressions reduces the amount of unnecessary technical information, such as timestamps, identifiers, and log levels, which simplifies the structure of messages and facilitates further vectoring.

It has been proven that preprocessing reduces the time required for vectorisation and model training by almost five times without losing accuracy. The accuracy of the model is not only preserved, but also improved: an increase in the F1-score and Matthews correlation coefficient (MCC) was recorded, which indicates an improvement in the quality of classification even in the presence of unbalanced classes. In addition, it was found that the use of TF-IDF can effectively emphasise meaningful words of diagnostic value, while suppressing repetitive terms. This has a positive effect on the model's noise resistance. It was also confirmed that the Log Loss value was decreasing, which indicates an increase in the model's confidence in its forecasts.

In the course of the study, a number of logically consistent and effective actions were identified that can be combined into a single pipeline for automatic processing of log files. Such a pipeline can be integrated into automated testing systems without significant losses for training the model, due to the effective allocation of the necessary information. This reduces the burden on analysts, ensures high classification accuracy, and reduces the time required to process large amounts of log data. It is important to note that the proposed approach does not depend on the use of complex or resource-intensive models and can be applied in real-world conditions without the need for high-performance equipment.

Thus, the results of the study confirmed the hypothesis that high-quality preprocessing of log files is a key factor in both speeding up calculations and improving the accuracy of models. The proposed approach can be successfully integrated into automated testing pipelines, helping to improve the efficiency of detecting and classifying software defects. In addition to its direct application value, the study highlights the potential of using simple word processing tools to solve complex software testing problems. Although the study was limited to one data set and one model type, the results provide prerequisites for further study of the method's stability to different domains and model architecture variants. The question of the extent to which other sources of information (e.g., contextual metadata) can influence the results of defect classification in logs remains relevant. In this context, the proposed solution is considered as a basis for future research.

Acknowledgements

The author is grateful to supervisor for support and valuable advice during the research process, as well as to colleagues who assisted in data analysis. Special thanks to the developers of the open data sets and libraries used in the study.

Funding

The study was not funded by any third-party organisation that could have influenced its results or conclusions.

Conflict of Interest

The author declares that there is no conflict of interest in the publication of this article.

References

[1] Abdalla, R.R., & Jumaa, A.K. (2022). Log file analysis based on machine learning: A survey. *UHD Journal of Science and Technology*, 6(2), 77-84. doi: 10.21928/uhdjst.v6n2y2022.pp77-84.

[2] Aljabri, M., Alahmadi, A.A., Mohammad, R.M.A., Aboulnour, M., Alomari, D.M., & Almotiri, S.H. (2022). Classification of firewall log data using multiclass machine learning models. *Electronics*, 11(12), article number 1851. doi: 10.3390/electronics11121851.

[3] BaseEstimator section in scikit-learn official documentation. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html.

[4] Bhanage, D.A., & Pawar, A.V. (2023). Improving classification-based log analysis using vectorisation techniques. In A.B. Reddy, S. Nagini, V.E. Balas & K.S. Raju (Eds.), *Proceedings of 3rd international conference on advances in computer engineering and communication systems. Lecture notes in networks and systems* (Vol. 612, pp. 271-282). Singapore: Springer. doi: 10.1007/978-981-19-9228-5_24.

[5] Brandão, A., & Georgieva, P. (2020). Log files analysis for network intrusion detection. In *Proceedings of the 2020 IEEE 10th international conference on intelligent systems (IS)* (pp. 328-333). Varna: IEEE. doi: 10.1109/IS48319.2020.9199976.

[6] Cao, C., Chicco, D., & Hoffman, M.M. (2020). The MCC-F1 curve: A performance evaluation technique for binary classification. *ArXiv*. doi: 10.48550/arXiv.2006.11278.

[7] Developing scikit-learn estimators. (n.d.). Retrieved from https://scikit-learn.org/stable/developers/develop.html.

[8] Géron, A. (2022). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems* (3rd ed.). Sebastopol: O'Reilly Media.

[9] HDFS architecture guide. (n.d.). Retrieved from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[10] Huangfu, Y. (2022). *Machine learning for log data analysis*. (PhD thesis, McMaster University, Hamilton, Canada).

[11] Hussein, S.A., & Répás, S.R. (2024). Anomaly detection in log files based on machine learning techniques. *Journal of Electrical Systems*, 20(3), 1299-1311. doi: 10.52783/jes.1505.

[12] Johnphill, O., Sadiq, A.S., Kaiwartya, O., & Aljaidi, M. (2024). An intelligent approach to automated operating systems log analysis for enhanced security. *Information*, 15(10), article number 657. doi: 10.3390/info15100657.

[13] Kapitanets, S., & Radelchuk, G. (2022). Peculiarities of data logging and influence of application type on the choice of logging methodology. *Bulletin of Khmelnytskyi National University*, 6(315), 98-101. doi: 10.31891/2307-5732-2022-315-6-98-101.

[14] Khan, Z.A., Shin, D., Bianculli, D., & Briand, L. (2024). Impact of log parsing on deep learning-based anomaly detection. *Empirical Software Engineering,* 29, article number 139. doi: 10.1007/s10664-024-10533-w.

[15] Khil, O., & Yakovina, V. (2023). Analysis of the problems of applying machine learning methods for evaluating and predicting software defects. *Scientific Bulletin of UNFU*, 33(3), 110-116. doi: 10.36930/40330316.

[16] Koval, V., Lysenko, V., Kiktev, N., Pylypenko, Yu., Samkov, O., Osinskiy, O., & Popov, I. (2022). Automated monitoring of time synchronisation devices and digital processing of vector measurements of dynamic characteristics of smart grid power systems. *Machinery & Energetics*, 13(2), 73-82. doi: 10.31548/machenergy.13(2).2022.73-82.

[17] Marjai, P., & Kiss, A. (2024). The usage of template mining in log file classification. *IEEE Access*, 12, 96378-96386. doi: 10.1109/ACCESS.2024.3426959.

[18] Meteriz, U., Yıldıran, F.N., Kim, J., & Mohaisen, D. (2020). Understanding the potential risks of sharing elevation information on fitness applications. In *40th international conference on distributed computing systems (ICDCS)* (pp. 464-473). Singapore: IEEE. doi: 10.1109/ICDCS47774.2020.00063.

[19] Peronto, R. (2024). The state of log data: 6 trends impacting observability and security. *Chronosphere Blog.* Retrieved from https://chronosphere.io/learn/observability-log-data-trends/.

[20] Prodeus, M., Nicheporuk, A., & Ivanchenko, O. (2024). The influence of data preprocessing on the performance of the random forest model in detecting network attacks. *Measuring and Computing Devices in Technological Processes*, 4, 415-419. doi: 10.31891/2219-9365-2024-80-51.

[21] Qin, Q., Aghili, R., Li, H., & Merlo, E. (2024). Preprocessing is all you need: Boosting the performance of log parsers with a general preprocessing framework. In *2025 IEEE international conference on software analysis, evolution, and reengineering (SANER)* (pp. 1-11). Singapore: IEEE. doi: 10.48550/arXiv.2412.05254.

[22] Ramachandran, S., Agrahari, R., Mudgal, P., Bhilwaria, H., Long, G., & Kumar, A. (2023). Automated log classification using deep learning. *Procedia Computer Science*, 218, 1722-1732. doi: 10.1016/j.procs.2023.01.150.

[23] Ryciak, P., Najgebauer, A., & Sołtysiak, M. (2022). A lightweight hybrid deep learning approach for log anomaly detection. *Applied Sciences*, 12(10), article number 5089. doi: 10.3390/app12105089.

[24] Salton, G., Wong, A., & Yang, C.S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11), 613-620. doi: 10.1145/361219.361220.

[25] Sandhu, A., & Mohammed, S. (2022). Detecting anomalies in logs by combining NLP features with embedding or TF-IDF, 1-5. doi: 10.36227/techrxiv.19498769.v1.

[26] Shirzad, E., & Saadatfar, H. (2022). Job failure prediction in Hadoop based on log file analysis. *International Journal of Computers and Applications*, 44(3), 260-269. doi: 10.1080/1206212X.2020.1732081.

[27] TfidfVectorizer. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[28] TransformerMixin. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html.

[29] Wang, J., Tang, Y., He, S., Zhao, C., Sharma, P.K., Alfarraj, O., & Tolba, A. (2020). LogEvent2vec: LogEvent-to-vector based anomaly detection for large-scale logs in Internet of Things. *Sensors*, 20(9), article number 2451. doi: 10.3390/s20092451.

[30] Zhou, J., Chen, Z., Wang, J., Zheng, Z., & Lyu, M.R. (2014). Trace bench: An open data set for trace-oriented monitoring. In *6th IEEE international conference on cloud computing technology and science* (pp. 519-526). Singapore: IEEE. doi: 10.1109/CloudCom.2014.79.

[31] Zhu, J., He, S., He, P., Liu, J., & Lyu, M.R. (2023). Loghub: A large collection of system log datasets for AI-driven log analytics. *ArXiv*. doi: 10.48550/arXiv.2008.06448.

Вплив обробки лог-файлів на швидкість навчання та точність класифікації дефектів

Антон Каяфюк

Аспірант
Київський національний університет технологій та дизайну
01011, вул. Мала Шияновська, 2, м. Київ, Україна
https://orcid.org/0009-0003-9917-0834

Анотація. Метою було дослідити вплив попередньої обробки лог-файлів автоматизованого тестування на швидкість векторизації та навчання моделей машинного навчання. Використано набір HDFS_v3_TraceBench, що містить понад 370 тисяч трасувань, зібраних у середовищі Hadoop Distributed File System. Обробка включала видалення шуму, лематизацію та зменшення дублікатів. Дані векторизовано методом Term frequency – inverse document frequency, після чого навчено модель RandomForestClassifier. Результати експериментів показали, що оптимізація вхідних даних дозволила зменшити загальний час обробки майже вп'ятеро. Час, необхідний для векторизації тексту та навчання моделі, скоротився, що дає змогу пришвидшити роботу з великими обсягами логів. При цьому точність класифікації не лише збереглася, а й продемонструвала незначне покращення: показники F1-score та коефіцієнта кореляції Метьюса залишилися стабільно високими. Також спостерігалося зниження значення Log Loss, що свідчило про підвищення впевненості моделі у власних прогнозах. Це особливо важливо в умовах незбалансованих класів, характерних для задач класифікації дефектів. Детальний аналіз виявив, що значна частина службової та повторюваної інформації в логах не є критичною для навчання моделі, а її видалення навпаки покращує якість підготовки даних. У ході роботи також було підтверджено, що отримані цільові мітки для логів відповідають типовим класам помилок. Реалізована обробка лог-файлів не лише скорочує обчислювальні витрати, але й підтримує або покращує якість прогнозування. Ці результати підтвердили доцільність включення етапу очищення та оптимізації логів у загальний процес побудови моделей машинного навчання для автоматизованого тестування. Отримані результати можуть бути інтегровані в автоматизовані пайплайни для класифікації дефектів і формування баг-репортів. Це сприятиме зменшенню обсягу ручної праці та підвищенню ефективності команд

Ключові слова: регулярні вирази; лематизація; векторизація; машинне навчання; автоматизація тестування