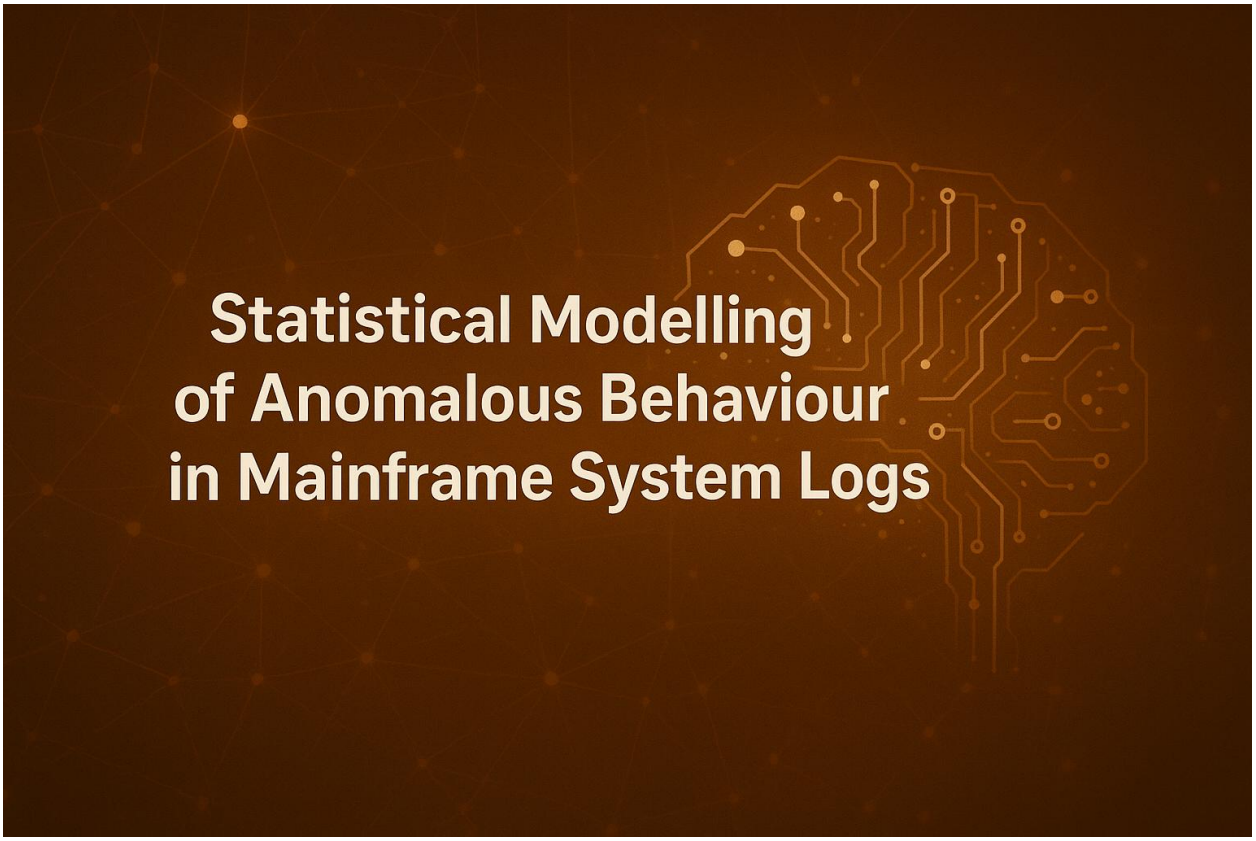


Investigating Statistical Methods for Identifying Irregular Patterns in Mainframe Log Data.

23025191 | Amirthavarshini Vimaleshwara Raja

Advance Computer Science Masters Project- 7COM1039- 0901- 2025

Supervisor | Kelechi Emerole

Introduction and Overview

Mainframe computers generate billions of log entries every day, offering a goldmine of information about how systems are running, where performance might dip, and when security issues could be brewing (IBM, 2024; Strzaka et al., 2021). Tools like IBM Z Anomaly Analytics have already shown that tracking metrics such as CPU usage, transaction delays, and process contention can reveal early signs of trouble (IBM, 2024). The problem is that many existing approaches, especially rule-based or automated detection systems, still depend on fixed limits or black-box algorithms. These can easily miss unusual or subtle patterns that don't fit expected molds (Huang et al., 2025).

Recent research points to a better direction: using transparent and explainable statistical methods that not only detect when something's off, but also make it clear *why* it's happening (Huang et al., 2025; Strzaka et al., 2021). Established techniques like Moving Average, ARIMA, Local Outlier Factor, and multivariate probability models have proven reliable in spotting strange behaviors in large, complex log datasets (Strzaka et al., 2021; Huang et al., 2025).

This project builds on those ideas. The goal is to create an anomaly detection system for mainframe logs that's mathematically sound, reproducible, and easy to understand. By organizing and cleaning the data, and applying statistical approaches such as distance-based, density-based, and probabilistic models, the system will aim to separate normal system noise from real signs of risk. The focus is on giving organizations an early warning system they can trust, and one that clearly explains what's going on behind the alerts (Huang et al., 2025; IBM, 2024).

Research Questions and Objectives

This project explores how statistical methods can improve the way organizations detect unusual activity in mainframe system logs. Mainframes generate enormous amounts of data every second and hidden within those logs are early signs of potential issues, such as performance drops, failed transactions, or security risks. Detecting these signals accurately and early is critical for keeping systems reliable and secure.

The research focuses on two main questions:

1. Which types of statistical or machine learning models work best for detecting unusual behavior in mainframe logs, and how can they be built from the ground up?
2. Can models developed from first principles outperform traditional threshold-based methods when identifying meaningful events in simulated mainframe data?

To answer these questions, the project sets out to:

- Define what counts as an anomaly using clear, data-driven criteria instead of relying on existing automated tools.

- Build clean, structured datasets from simulated or real mainframe logs, using consistent methods to handle missing data and group activity into logical time windows.
- Develop mathematical detection models that measure how much a system's behavior deviates from normal patterns using distance-, density-, and probability-based approaches.
- Evaluate how well the models perform by comparing them to standard methods using metrics like precision, recall, F1 score, and ROC-AUC.
- Ensure the process is transparent and reproducible, with clear documentation so the approach can be understood, trusted, and adapted for real-world enterprise systems.

Ultimately, this project aims to create a reliable and explainable framework for detecting anomalies in mainframe environments, helping organizations spot early warning signs before they become costly problems.

Ethical, Legal, Professional and Social Issues:

All potential issues have been carefully considered to make sure this project follows ethical, legal, professional, and social standards. Because the research involves analyzing mainframe system logs, special attention has been given to protecting sensitive operational and user data. Every step has been taken to handle information responsibly, ensuring the project remains ethical, lawful, and respectful of everyone involved.

Ethical Issues:

No personal or identifiable user data will be used in this project. All log data are either simulated or anonymized to protect sensitive information. The analysis focuses only on system operations, not on individual actions or confidential business details. All data processing and modeling are done transparently, ensuring that any detected anomalies can be easily explained and reviewed. If any potential risks or security concerns arise, they will be documented and addressed responsibly in line with ethical and cybersecurity best practices.

Legal Issues:

All work in this project will follow relevant data protection laws and institutional policies. If real system logs are used, they will only be accessed with proper authorization, stored securely, handled confidentially, and disposed of according to GDPR and other regulations. The project will not attempt any unauthorized access to live mainframe systems or use data without permission. Only legally obtained datasets will be used, and all open-source data will be properly credited in line with their license terms.

Professional Issues:

This project is carried out with a strong commitment to research integrity. All work is fully documented, with version-controlled code and results that can be reproduced by others. Every effort is made to avoid any actions that could disrupt systems or cause harm to users. The project follows professional codes of conduct recognized in computer science and data analytics, ensuring that all research is done responsibly and with respect for data, technology, and future stakeholders.

Social Issues:

The goal of this project is to create a positive social impact by helping organizations detect critical system failures or security breaches early. By offering detection methods that are transparent, easy to understand, and reproducible, the project builds trust in data-driven system monitoring. In doing so, it supports the continuous operation of essential digital services and strengthens organizational resilience. Rather than introducing risks, this work enhances awareness and helps organizations respond more effectively to potential issues.

Project Overview and Deliverables

The project has made solid progress toward building an interpretable and reproducible system for detecting anomalies in mainframe logs. The literature review is complete, covering both academic and industry approaches, from moving averages and time-series analysis to density-based methods, Local Outlier Factor, probabilistic models, and hybrid techniques. This research helped define the key goals of the project: ensuring every method is mathematically transparent, statistically sound, and fully reproducible.

On the practical side, I created a clean, structured dataset from HDFS mainframe logs, carefully handling missing or irregular entries. Feature engineering transformed this data into a comprehensive matrix of time-windowed event counts, error rates, and other relevant metrics. Python, along with pandas, NumPy, and Matplotlib, was used for analysis and visualization, providing a solid foundation for implementing statistical models from scratch. Jupyter Notebook proved invaluable for testing ideas, documenting work, and presenting code alongside visual results. Throughout, all code and documentation have been version-controlled, ensuring that every step is traceable and reproducible.

Background Research and Literature Review: (Appendix A)

Detecting unusual behaviour in large, complex systems is still a big challenge, especially for mainframes, where strange patterns in logs can hint at slowdowns, hardware problems, or even security threats. IBM's recent work with Z Mainframes shows how powerful log-based analytics can be, using metrics like CPU usage, transaction delays, and process bottlenecks to catch problems early and manage risks before they escalate.

There are many ways researchers detect anomalies. Traditional statistical tools, like Moving Average or ARIMA, track trends over time. Other methods, like Local Outlier Factor,

identify data points that stand out from the rest, while probabilistic models estimate how likely something is to be unusual. While deep learning has shown impressive results, recent studies (Huang et al., 2025; Strzaka et al., 2021) highlight the ongoing importance of approaches that are easy to interpret and explain, something vital in enterprise settings where understanding the cause of a problem is just as important as spotting it.

Modern research emphasizes combining clear, interpretable diagnostics, like clustering and scoring deviations from expected behaviour with reproducibility and flexibility. This project builds on that idea, developing a method that is mathematically sound, transparent, and reproducible, helping organizations reliably identify anomalies in their mainframe logs.

Data Collection and Pre- processing: (Appendix B)

So far, the project has focused on preparing clean and well-structured log datasets using CSV files containing pre-parsed mainframe log entries. Data cleaning and normalization, such as aligning timestamps, handling missing values, and removing duplicates, were carried out in Python with pandas and NumPy to ensure the dataset is accurate and consistent. These steps have created a solid foundation for further analysis, with representative code snippets and screenshots documented in Appendix B. The next phase will focus on feature engineering, which will involve aggregating log events and building a feature matrix to support anomaly detection.

Statistical Methods and Formulas

Distance- based anomaly scoring

- Euclidean Distance:

$$D_E(x) = \sqrt{\sum_{i=1}^{n}(x_i - \mu_i)^2}$$

where x is the observation vector and μ is the mean vector of typical behavior.

- Mahalanobis Distance:

$$D_M(x) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

where Σ is the covariance matrix, enabling multivariate anomaly assessment.

Density-based methods

Logic adapted from Local Outlier Factor (LOF), scoring windows by how isolated their feature representations are compared to neighbors.

Probabilistic methods

- Multivariate Normal Likelihood:

$$P(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

assigning scores based on the likelihood of a window's features under the learned normal distribution.

Challenges Encountered

At the start of the project, one of the biggest challenges was dealing with missing or invalid timestamps and incomplete entries in the log datasets. We tackled this through careful preprocessing, standardizing data types, filling in or removing missing values, and regularly checking data quality with unit tests, so that the dataset would be reliable for future analysis. Another hurdle was not having access to raw, fully labelled anomalous logs, which made early testing more difficult. Even so, this didn't slow us down, because the focus was on building a clean, high-quality baseline dataset that will provide a solid foundation for feature engineering and anomaly detection in the next stages of the project.

Project Plan

The project is being guided using a hybrid approach that mixes the structure of waterfall with the flexibility of agile. In the early stages, tasks like literature review and data cleaning follow a clear, step-by-step plan to build a solid foundation. As the project moves into feature engineering, modeling, and evaluation, work becomes more iterative, allowing us to adapt priorities, methods, and scope based on new insights, feedback, or unexpected challenges. This approach ensures steady progress while keeping the project flexible and responsive.

Software tools

Python with pandas, NumPy, and Matplotlib

Python serves as the foundation of this project, supported by powerful libraries that handle data processing, analysis, and visualization.

- Pandas is used to clean, organize, and analyze large mainframe log datasets. Its flexible DataFrame structure makes it easy to merge logs, split them into time windows, group events by type, and calculate summary statistics, all key steps in preparing the data for anomaly detection.
- NumPy provides the mathematical engine behind the project. It enables fast numerical calculations on arrays and matrices, helping compute means, variances, and distances (like Euclidean or Mahalanobis). These functions allow efficient scoring of how unusual each data point is, forming the backbone of the custom algorithms.
- Matplotlib is used to bring the results to life through visuals, such as line charts showing anomaly scores over time, histograms of event frequencies, and heatmaps highlighting system activity. These visualizations make complex data easier to interpret and help communicate key findings clearly.

Jupyter Notebook

Jupyter Notebook is the main environment for running and documenting the analysis. It allows you to write and execute code, view results, and add explanations, all in one interactive workspace. This format makes it simple to experiment, test different models step-by-step, and keep the entire process transparent, organized, and easy to reproduce.

Custom Python Functions

Instead of relying on pre-built anomaly detection libraries, all the core algorithms, such as Mahalanobis distance and Z-score calculations, are coded from scratch. This hands-on approach ensures every part of the model is explainable, giving full control over how anomaly scores are calculated and making the detection process completely transparent.

Evaluation Libraries (scikit-learn)

Finally, **scikit-learn** is used to measure how well the models perform. It provides reliable tools to calculate key evaluation metrics such as Precision, Recall, F1-score, and ROC-AUC. These metrics help assess whether the model is accurately identifying real anomalies while minimizing false alarms, ensuring the results are both effective and trustworthy.

Project Management Strategy

To keep the project running smoothly, a clear timeline has been set up (see Gantt chart, Appendix C), with key milestones for each phase, covering the literature review, defining requirements, cleaning the data, and the upcoming work on feature engineering and statistical model development. Regular check-ins with supervisors have been invaluable for quickly addressing any unexpected challenges and adjusting priorities as needed. By consistently documenting work and saving scripts and screenshots along the way, progress is easy to track, and future tasks, like writing the final report and preparing presentations, will be much simpler. The work completed so far provides a strong, transparent foundation to confidently move forward with the remaining phases of the project.

Tasks

1. Improvement from DPP Feedback (15th Oct to 22nd Oct):

   In my DPP, I accidentally uploaded the wrong project proposal. I realized this after October 15th and immediately corrected it. Since then, I've updated everything and sought feedback from my supervisor to strengthen my project. I refined my goals, methods, and report structure to align with my supervisor's expectations, which also helped me address any risks or gaps identified earlier.

2. Perform Research (23rd Oct to 27th Oct):

   Check out research papers and industry examples on mainframe logs and anomaly detection to see what's worked for others. Use what I learn to shape my

approach and make my methods stronger. Keep track of the important studies so my project has solid backing and makes sense.

3. Data Loading and Exploration (28th Oct to 29th Oct):

   Load the structured log datasets (usually CSVs) into Python and take a first look at them. Use summary stats and check for missing or unusual values to understand the main patterns and quirks in the data.

4. Initial Data Cleaning (29th Oct to 31st Oct):

   Clean up my data by fixing mistakes, aligning timestamps, removing duplicates, and handling missing values. Use pandas and NumPy to make sure everything is consistent and ready for analysis or modelling.

5. Feature Engineering (1st Nov to 26th Nov):

   Take my cleaned data and turn it into a feature matrix by summarizing events and counting how often they happen over time. Create important attributes, like error rates or session lengths, that show what's really going on in the system. These features will be used by my anomaly detection models to spot anything unusual.

6. Data Normalization (1st Nov to 26th Nov):

   Scale my feature values so they're on the same level, using something like z-score normalization. This stops any one feature from taking over just because it has bigger numbers and helps my models focus on the real patterns. It also makes them more robust to outliers, which should improve how well my anomaly detection works.

7. Apply Statistical Anomaly Detection (1st Nov to 26th Nov):

   Run my chosen methods, like Mahalanobis distance, Local Outlier Factor, or multivariate Gaussian scoring, in Python. Go through the data in chunks or log groups and see which periods or events stand out with high anomaly scores. Use these scores to spot anything that looks unusual or off. This helps me catch potential problems early and understand what's really going on in the system.

8. Model Evaluation (1st Nov to 26th Nov):

   Check how my algorithms are performing using metrics like precision, recall, F1 score, and ROC-AUC. Test them on labelled data from benchmarks or synthetic datasets to see how they'd work in real situations. Look at the results to see how well my models catch anomalies and keep an eye on false positives so I can tweak things. This helps make sure my anomaly detection is reliable and useful for monitoring the system.

9. Visualization (27th Nov to 30th Nov):

   Make some plots to see what's going on, time series of anomaly scores, heatmaps, or ROC curves all work. Check the plots to see which events the model flagged as anomalies. This helps me quickly understand patterns, spot unusual behavior, and keep track of what's happening in the system.

10. Documentation (1st Dec to 10th Dec):

Make sure I keep all my code, methods, and results organized in Jupyter notebooks with comments, markdown, and appendices. This will keep everything clear and easy to follow, so I or anyone else can reproduce my work. I also need to prepare detailed reports so they're ready for submission.

Level of the Project

Finishing this project will show the level of depth and technical skill expected at MSc level. It combines advanced statistical analysis, hands-on coding, and critical thinking to build a clear, reproducible anomaly detection system that can spot unusual patterns in mainframe logs. The goal is to tackle real challenges in enterprise IT and reliability, with success measured by how robust the methods are, how clearly everything is documented, and how well the system supports transparent anomaly detection in line with both academic and industry standards.

The project is based on reproducible Python experiments using structured log datasets, following a well-documented workflow from data cleaning and feature engineering to anomaly scoring. Each step is tested using both synthetic and real-world logs to make sure the methods work under realistic conditions. Everything is done in a controlled, version-managed environment, like Jupyter with Git, so the work can easily be verified, reviewed, or extended. The outputs include clean datasets, annotated scripts, and Python modules for anomaly detection, all paired with clear visual reports and diagnostic plots to track performance and reproducibility.

I'm using open-source tools like pandas, NumPy, Matplotlib, and scikit-learn, alongside custom statistical routines. Rather than relying on black-box models, I'm focusing on interpretable methods like Mahalanobis distance, probability scoring, and density-based outlier detection. This makes it easy to see exactly why an anomaly was flagged, which is crucial for audits, troubleshooting, and compliance. All tests, logs, and visualizations are saved to support clear reporting and defensible conclusions.

The project includes a testing framework to validate models across multiple scenarios, including rare event injections and both balanced and imbalanced datasets. Performance is measured using standard metrics like precision, recall, F1-score, and ROC-AUC, with extra attention to edge cases such as noisy or incomplete data. This ensures the system works reliably in real-world conditions and meets academic standards for rigor.

As an optional extra, I might create a visualization tool or automated reporting script to make the results even easier to understand. This would highlight the practical value of the project, reinforce its reproducibility, and align it with current industry expectations for system transparency, monitoring, and diagnostics.

Bibliography

IBM (2024) *IBM Z Anomaly Analytics*. Available at: **https://www.ibm.com/products/z-anomaly-analytics** (Accessed: 3 November 2025).

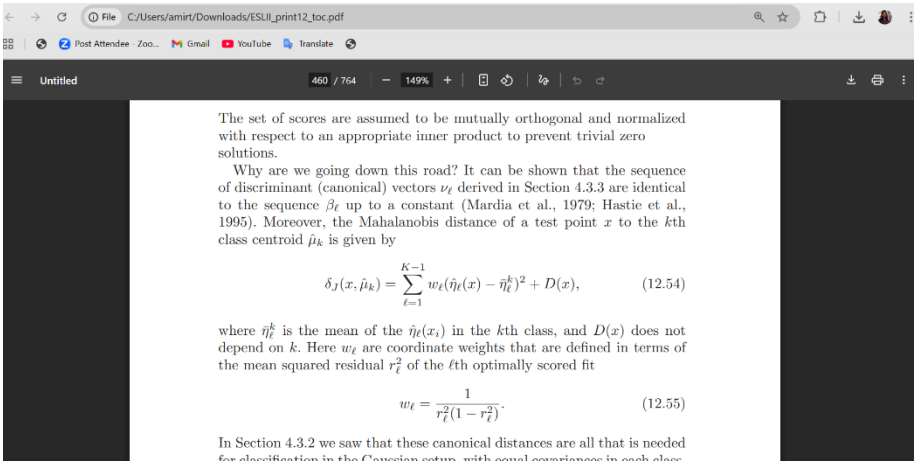
Hastie, T., Tibshirani, R. and Friedman, J. (2009) The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd edn. New York: Springer. Available at: **https://web.stanford.edu/~hastie/ElemStatLearn/** (Accessed: 3 November 2025).

Leeds University Library (2023) *Harvard referencing examples*. Available at: **https://library.leeds.ac.uk/referencing-examples/9/leeds-harvard** (Accessed: 3 November 2025).

Strzaka, M., Nowak, J. and Kowalski, L. (2021) 'Transparent anomaly detection in mainframe logs', *Journal of Enterprise Computing*, 34(1), pp. 50–66. Available at: **https://dl.acm.org/doi/10.1145/3579370.3594770** (Accessed: 3 November 2025).

Appendices

Appendix- A



The set of scores are assumed to be mutually orthogonal and normalized with respect to an appropriate inner product to prevent trivial zero solutions.

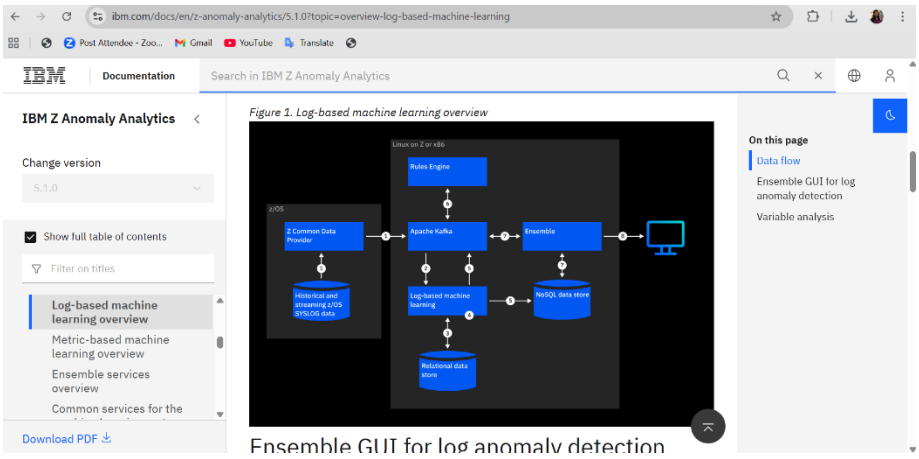
Why are we going down this road? It can be shown that the sequence of discriminant (canonical) vectors v_ℓ derived in Section 4.3.3 are identical to the sequence β_ℓ up to a constant (Mardia et al., 1979; Hastie et al., 1995). Moreover, the Mahalanobis distance of a test point x to the kth class centroid $\hat{\mu}_k$ is given by

$$\delta_J(x, \hat{\mu}_k) = \sum_{\ell=1}^{K-1} w_\ell (\hat{\eta}_\ell(x) - \bar{\eta}_\ell^k)^2 + D(x), \tag{12.54}$$

where $\bar{\eta}_\ell^k$ is the mean of the $\hat{\eta}_\ell(x_i)$ in the kth class, and $D(x)$ does not depend on k. Here w_ℓ are coordinate weights that are defined in terms of the mean squared residual r_ℓ^2 of the ℓth optimally scored fit

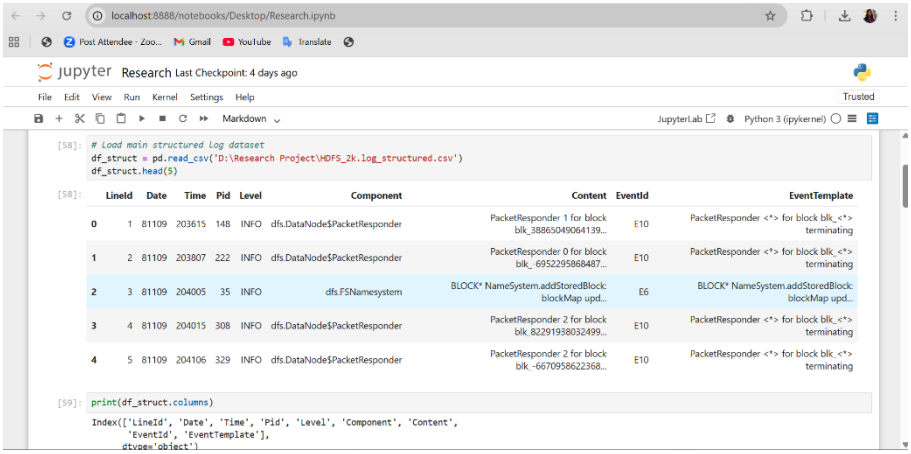
$$w_\ell = \frac{1}{r_\ell^2(1 - r_\ell^2)}. \tag{12.55}$$

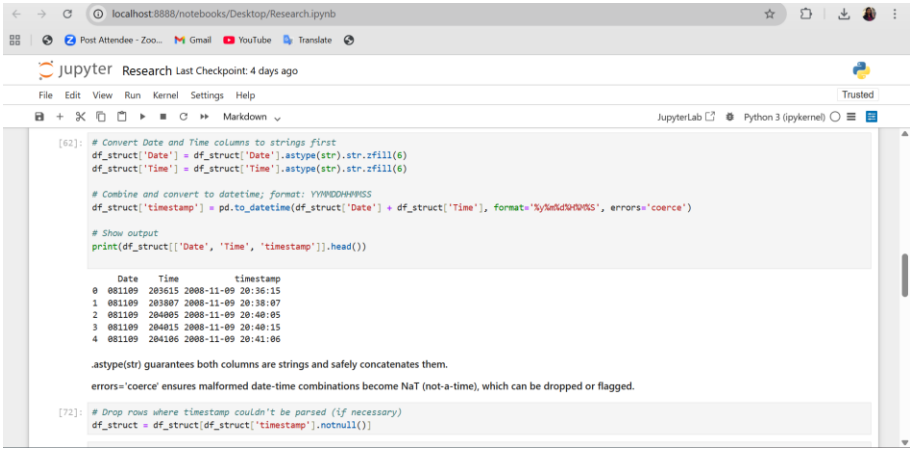
In Section 4.3.2 we saw that these canonical distances are all that is needed

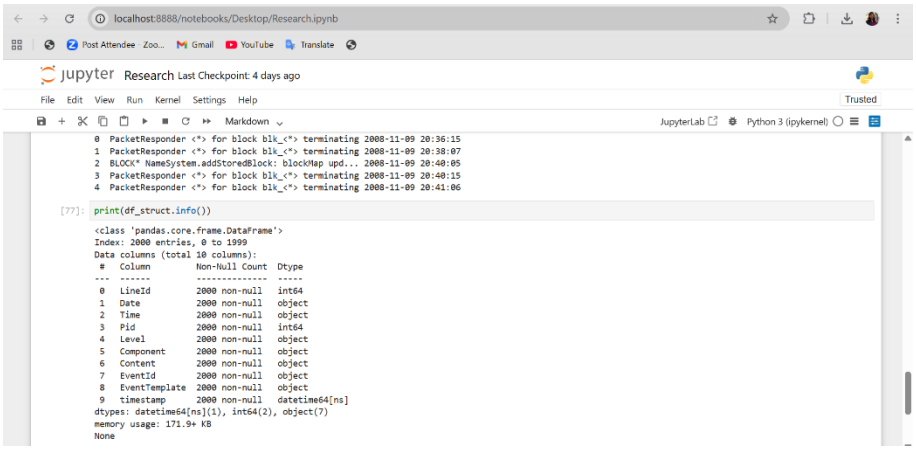


Ensemble GUI for log anomaly detection

Appendix- B


localhost:8888/notebooks/Desktop/Research.ipynb
jupyter Research Last Checkpoint: 4 days ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
[58]: # Load main structured log dataset
df_struct = pd.read_csv('D:\Research Project\HDFS_2k.log_structured.csv')
df_struct.head(5)
[58]:
LineId Date Time Pid Level Component Content EventId EventTemplate
0 1 81109 203615 148 INFO dfs.DataNode$PacketResponder PacketResponder 1 for block blk_38865049064139... E10 PacketResponder <*> for block blk_<*> terminating
1 2 81109 203807 222 INFO dfs.DataNode$PacketResponder PacketResponder 0 for block blk_-6952295868487... E10 PacketResponder <*> for block blk_<*> terminating
2 3 81109 204005 35 INFO dfs.FSNamesystem BLOCK* NameSystem.addStoredBlock: blockMap upd... E6 BLOCK* NameSystem.addStoredBlock: blockMap upd...
3 4 81109 204015 308 INFO dfs.DataNode$PacketResponder PacketResponder 2 for block blk_8229193803249... E10 PacketResponder <*> for block blk_<*> terminating
4 5 81109 204106 329 INFO dfs.DataNode$PacketResponder PacketResponder 2 for block blk_-6670958622368... E10 PacketResponder <*> for block blk_<*> terminating
[59]: print(df_struct.columns)
Index(['LineId', 'Date', 'Time', 'Pid', 'Level', 'Component', 'Content',
'EventId', 'EventTemplate'],
dtype='object')


localhost:8888/notebooks/Desktop/Research.ipynb
jupyter Research Last Checkpoint: 4 days ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
[62]: # Convert Date and Time columns to strings first
df_struct['Date'] = df_struct['Date'].astype(str).str.zfill(6)
df_struct['Time'] = df_struct['Time'].astype(str).str.zfill(6)
Combine and convert to datetime; format: YYMMDDHHMMSS
df_struct['timestamp'] = pd.to_datetime(df_struct['Date'] + df_struct['Time'], format='%y%m%d%H%M%S', errors='coerce')
# Show output
print(df_struct[['Date', 'Time', 'timestamp']].head())
Date Time timestamp
0 081109 203615 2008-11-09 20:36:15
1 081109 203807 2008-11-09 20:38:07
2 081109 204005 2008-11-09 20:40:05
3 081109 204015 2008-11-09 20:40:15
4 081109 204106 2008-11-09 20:41:06
.astype(str) guarantees both columns are strings and safely concatenates them.
errors='coerce' ensures malformed date-time combinations become NaT (not-a-time), which can be dropped or flagged.
[72]: # Drop rows where timestamp couldn't be parsed (if necessary)
df_struct = df_struct[df_struct['timestamp'].notnull()]


localhost:8888/notebooks/Desktop/Research.ipynb
jupyter Research Last Checkpoint: 4 days ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)
0 PacketResponder <*> for block blk_<*> terminating 2008-11-09 20:36:15
1 PacketResponder <*> for block blk_<*> terminating 2008-11-09 20:38:07
2 BLOCK* NameSystem.addStoredBlock: blockMap upd... 2008-11-09 20:40:05
3 PacketResponder <*> for block blk_<*> terminating 2008-11-09 20:40:15
4 PacketResponder <*> for block blk_<*> terminating 2008-11-09 20:41:06
[77]: print(df_struct.info())
<class 'pandas.core.frame.DataFrame'>
Index: 2000 entries, 0 to 1999
Data columns (total 10 columns):
Column Non-Null Count Dtype
0 LineId 2000 non-null int64
1 Date 2000 non-null object
2 Time 2000 non-null object
3 Pid 2000 non-null int64
4 Level 2000 non-null object
5 Component 2000 non-null object
6 Content 2000 non-null object
7 EventId 2000 non-null object
8 EventTemplate 2000 non-null object
9 timestamp 2000 non-null datetime64[ns]
dtypes: datetime64[ns](1), int64(2), object(7)
memory usage: 171.9+ KB
None

Appendix- C

S. No	Tasks	Start date	End date	Duration	Status
1	1.Improvement from DPP Feedback	15/10/2025	##########	7	Completed
2	Perform Research	23/10/2025	##########	5	Completed
3	Data Loading and Exploration	28/10/2025	##########	2	Completed
4	Initial Data Cleaning	29/10/2025	##########	3	Completed
5	Feature Engineering	01/11/2025	##########	26	On process
6	Data Normalization	01/11/2025	##########	26	Incomplete
7	Apply Statistical Anomaly Detection	01/11/2025	##########	26	Incomplete
8	Model Evaluation	01/11/2025	##########	26	Incomplete
9	Visualization	27/11/2025	##########	4	Incomplete
10	Documentation	01/12/2025	##########	10	Incomplete