Log Anomaly Detection

on EuXFEL Nodes

Antonin Sulc, Annika Eichler, Tim Wilksen
Cape Town,

HELMHOLTZ

DESY.

Introduction

- **Log data** (LD) provides a detailed record of events, actions, and state changes.

Introduction

- **Log data** (LD) provides a detailed record of events, actions, and state changes.
- **Log anomaly detection** (LAD) identifies unusual patterns in LD that may indicate problems or issues.

Introduction

- **Log data** (LD) provides a detailed record of events, actions, and state changes.
- **Log anomaly detection** (LAD) identifies unusual patterns in LD that may indicate problems or issues.
- We show a **novel unsupervised log anomaly detection** approach tailored for the purpose of European XFEL watchdog logs using the sequential nature of the log messages.

Introduction

- **Log data** (LD) provides a detailed record of events, actions, and state changes.
- **Log anomaly detection** (LAD) identifies unusual patterns in LD that may indicate problems or issues.
- We show a **novel unsupervised log anomaly detection** approach tailored for the purpose of European XFEL watchdog logs using the sequential nature of the log messages.
- The EuXFEL watchdog log has some **features** which makes the problem challenging!

Existing Methods

- **Rule-based methods** rely on manually defined rules and patterns, which are labor-intensive and limited to predefined patterns.

Existing Methods

- **Rule-based methods** rely on manually defined rules and patterns, which are labor-intensive and limited to predefined patterns.
- **Supervised deep learning models** require large labeled datasets, which are expensive and not always available.

Existing Methods

- **Rule-based methods** rely on manually defined rules and patterns, which are labor-intensive and limited to predefined patterns.
- **Supervised deep learning models** require large labeled datasets, which are expensive and not always available.
- **Unsupervised methods** like clustering treat logs independently rather than sequentially, but missing contextual information.

Existing Methods

- **Rule-based methods** rely on manually defined rules and patterns, which are labor-intensive and limited to predefined patterns.
- **Supervised deep learning models** require large labeled datasets, which are expensive and not always available.
- **Unsupervised methods** like clustering treat logs independently rather than sequentially, but missing contextual information.

...and we show an approach which can help with above stated problems!

Approach - Steps

1 **Preprocessing and tokenization**

Approach - Steps

1 **Preprocessing and tokenization**

2 **Embedding**

Approach - Steps

1. **Preprocessing and tokenization**
2. **Embedding**
3. **Parameter Estimation**

Approach - Steps

1. Preprocessing and tokenization
2. Embedding
3. Parameter Estimation
4. Anomaly Detection

Approach - Steps

1 **Preprocessing and tokenization**

2 **Embedding**

3 **Parameter Estimation**

4 **Anomaly Detection**

```
from hmmlearn import hmm
import numpy as np

x = np.stack([[0,1],[1,0],[0,1],[1,0],[0,1],[1,0],[0,1],[1,0]])
model = hmm.GaussianHMM(n_components=2, covariance_type="diag")
model.fit(x[:-1,:])
logp = []
for i in range(1,x.shape[0]+1):
    logp.append(model.score(x[:i]))

logp = np.array(logp)
score = logp[:-1] - logp[1:]
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

```
A problem ### with the XFEL/DEVICE 235
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,

```
A problem with the XFELDEVICE 235
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,
2. Substitute all potential names with one special symbol,

```
A problem with the $name 235
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,
2. Substitute all potential names with one special symbol,
3. Numbers replaced with special symbols,

```
A problem with the $name $nz
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,
2. Substitute all potential names with one special symbol,
3. Numbers replaced with special symbols,
4. Log entry is changed to lower-case,

```
a problem with the $name $nz
```

Step 1 - Preprocessing and Tokenization

Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,
2. Substitute all potential names with one special symbol,
3. Numbers replaced with special symbols,
4. Log entry is changed to lower-case,
5. Tokenization.

(a, `problem`,`with`,`the`,`$name`,`$nz`)

Step 1 - Preprocessing and Tokenization

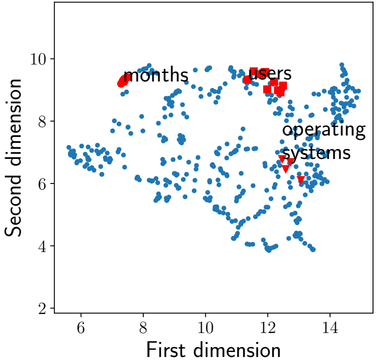
Some words appear only rarely, they should be eliminated (device names, numbers)

1. Filter special characters,
2. Substitute all potential names with one special symbol,
3. Numbers replaced with special symbols,
4. Log entry is changed to lower-case,
5. Tokenization.
6. English stop words are removed (`a`, `the`, ...),

(`problem`,`$name`,`$nz`)

Step 2 - Embedding

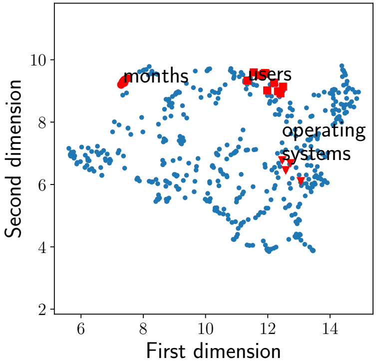
We need a log entry represented as a vector.



Step 2 - Embedding

We need a log entry represented as a vector.

- Word2Vec takes a word and represents it as a vector based on context (adjacent words).

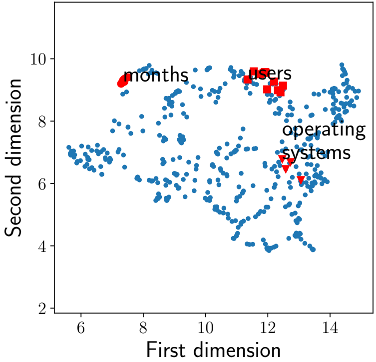


Step 2 - Embedding

We need a log entry represented as a vector.

- Word2Vec takes a word and represents it as a vector based on context (adjacent words).
- Semantically similar words are represented with vectors that point to a similar direction

  linear accelerator $\leftrightarrow$ linac



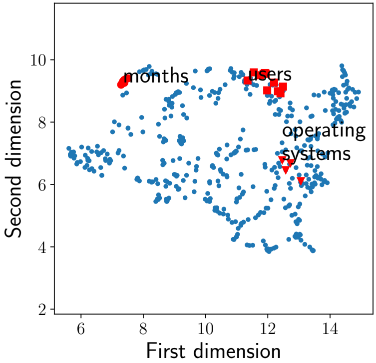
Step 2 - Embedding

We need a log entry represented as a vector.

- Word2Vec takes a word and represents it as a vector based on context (adjacent words).
- Semantically similar words are represented with vectors that point to a similar direction

  linear accelerator $\leftrightarrow$ linac

- A good feature of Word2Vec is also the ability to do basic arithmetic operations.



Step 2 - Embedding

We need a log entry represented as a vector.

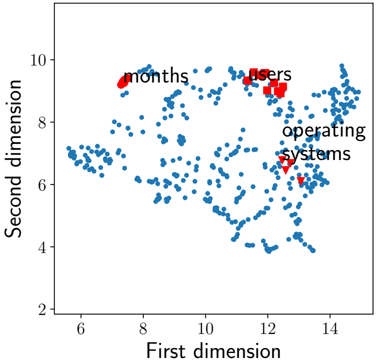
> Word2Vec takes a word and represents it as a vector based on context (adjacent words).

> Semantically similar words are represented with vectors that point to a similar direction
linear accelerator ↔ linac

> A good feature of Word2Vec is also the ability to do basic arithmetic operations.

To represent a log entry `problem,$name,$nz`:

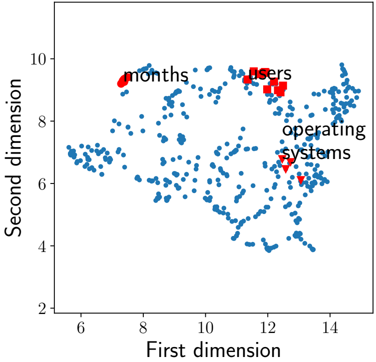
1 **Embedding** `[0.1,...]` `[2,...]` `[0.8,...]`
`problem` `$name` `$nz`



Step 2 - Embedding

We need a log entry represented as a vector.

- Word2Vec takes a word and represents it as a vector based on context (adjacent words).
- Semantically similar words are represented with vectors that point to a similar direction
  linear accelerator ↔ linac
- A good feature of Word2Vec is also the ability to do basic arithmetic operations.



To represent a log entry `problem,$name,$nz`:

1 **Embedding** `[0.1,...]` `[2,...]` `[0.8,...]`
`problem` `$name` `$nz`

2 **Summation** `[3.232,...]`

Our Approach - Step 3 - Modeling the Sequence

> HMM (Hidden Markov Model) learns distribution over likely (log) sequences (embedded with Word2Vec)

Our Approach - Step 3 - Modeling the Sequence

- HMM (Hidden Markov Model) learns distribution over likely (log) sequences (embedded with Word2Vec)
- Parameters are estimated from past log sequences.

Our Approach - Step 3 - Modeling the Sequence

- HMM (Hidden Markov Model) learns distribution over likely (log) sequences (embedded with Word2Vec)
- Parameters are estimated from past log sequences.
- A new log entry o_i is scored by

$$
\begin{aligned}
s_i &= \log \frac{\text{prob. of prev.logs}}{\text{prob. of prev.logs + new one}} \\
&= \log \frac{p_\theta(o_1,\ldots,o_{i-1})}{p_\theta(o_1,\ldots,o_i)} .
\end{aligned}
$$

Our Approach - Step 3 - Modeling the Sequence

- HMM (Hidden Markov Model) learns distribution over likely (log) sequences (embedded with Word2Vec)
- Parameters are estimated from past log sequences.
- A new log entry o_i is scored by

$$
\begin{aligned}
s_i &= \log \frac{\text{prob. of prev.logs}}{\text{prob. of prev.logs + new one}} \\
&= \log \frac{p_\theta(o_1,\ldots,o_{i-1})}{p_\theta(o_1,\ldots,o_i)} .
\end{aligned}
$$

- Low probability entries under learned HMM identified as anomalies.

Our Approach - Step 3 - Modeling the Sequence

- HMM (Hidden Markov Model) learns distribution over likely (log) sequences (embedded with Word2Vec)
- Parameters are estimated from past log sequences.
- A new log entry o_i is scored by

$$
\begin{aligned}
s_i &= \log \frac{\text{prob. of prev.logs}}{\text{prob. of prev.logs + new one}} \\
&= \log \frac{p_\theta(o_1, \ldots, o_{i-1})}{p_\theta(o_1, \ldots, o_i)}
\end{aligned}
.
$$

- Low probability entries under learned HMM identified as anomalies.
- Detects anomalies from disruptions of expected patterns.

Step 3 - Modeling the Sequence - Features

Unsupervised	**No labels** needed, pure sequence modeling
Novelty	Handles **novel entries** based on contextual irregularity
Data	**Minimal number of parameters**
Robustness	Capable of flagging anomalies even if some are in training logs
Easy	With proper packages **10 lines** of Python code

Tiny Example

(TEST,OK,)

Tiny Example

(TEST,OK,TEST,OK,)

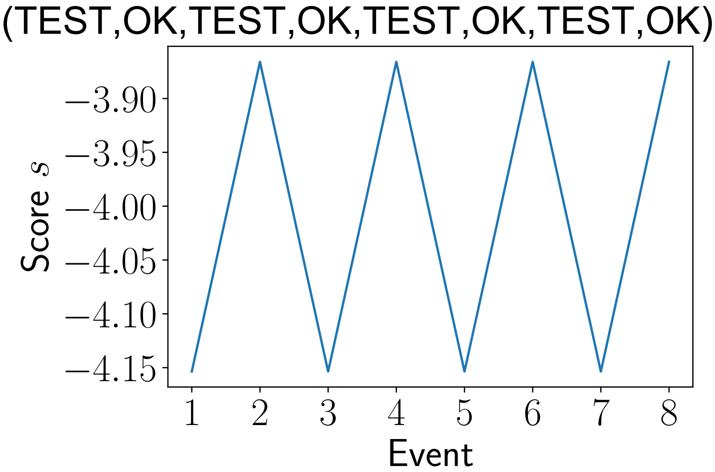
Tiny Example

(TEST,OK,TEST,OK,TEST,OK,   )

Tiny Example

(TEST,OK,TEST,OK,TEST,OK,TEST,OK)

Tiny Example



Tiny Example - Sequential Anomaly

(TEST,OK,)

Tiny Example - Sequential Anomaly

(TEST,OK,TEST,OK,   )

Tiny Example - Sequential Anomaly

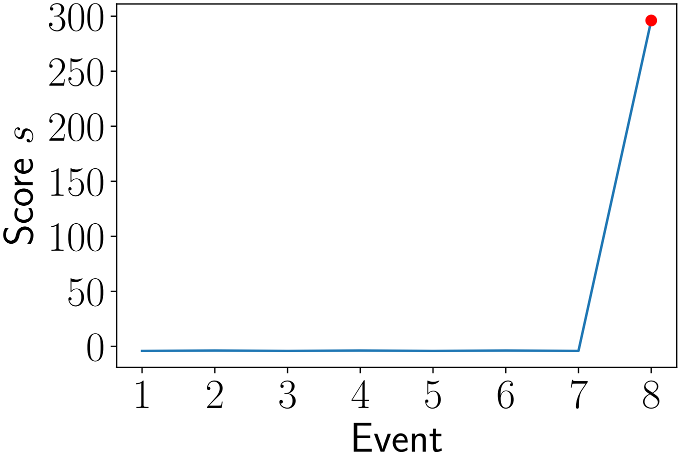
(TEST,OK,TEST,OK,TEST,OK,   )

Tiny Example - Sequential Anomaly

(TEST,OK,TEST,OK,TEST,OK,TEST,**TEST**)

Tiny Example - Sequential Anomaly

(TEST,OK,TEST,OK,TEST,OK,TEST,**TEST**)



Tiny Example - Unexpected Message Anomaly

(TEST,OK,   )

Tiny Example - Unexpected Message Anomaly

(TEST,OK,TEST,OK,)

Tiny Example - Unexpected Message Anomaly

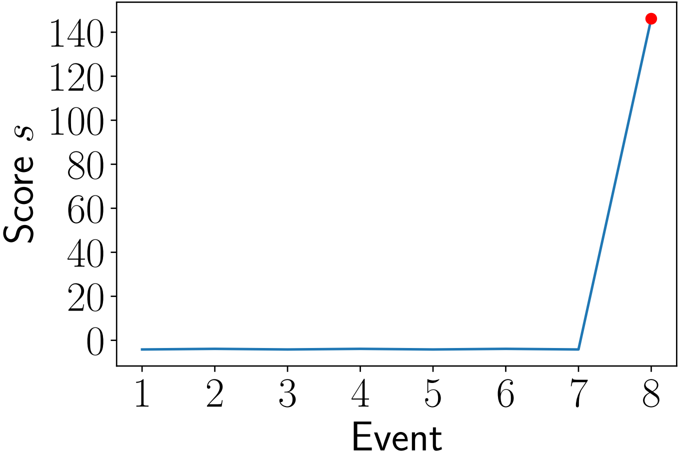
(TEST,OK,TEST,OK,TEST,OK,)

Tiny Example - Unexpected Message Anomaly

(TEST,OK,TEST,OK,TEST,OK,TEST,**ERROR**)

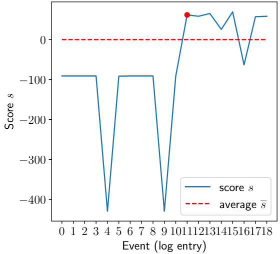
Tiny Example - Unexpected Message Anomaly

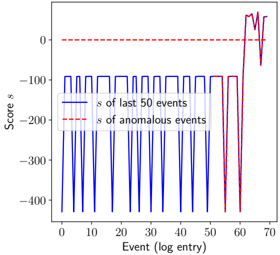
(TEST,OK,TEST,OK,TEST,OK,TEST,**ERROR**)



Example 1

```
                                     ⋮
0    remoteerrors errorcount $nz
1    remoteerrors errorcount $nz
2    remoteerrors errorcount $nz
3    remoteerrors errorcount $nz
4    remoteerrors errorcount $nz toggled $nz times $nz min
5    remoteerrors errorcount $nz
6    remoteerrors errorcount $nz
7    remoteerrors errorcount $nz
8    remoteerrors errorcount $nz
9    remoteerrors errorcount $nz toggled $nz times $nz min
10   remoteerrors errorcount $nz
11   rpccheck nullproc error
12   rpccheck fails $nz kill $nz
13   getpid pid not match process name
14   no process try start
15   getpid pid not match process name
16   pid change $nz $nz
17   rpccheck nullproc error
18   rpccheck fails $nz kill $nz
19   rpccheck fails $nz kill $nz
                                     ⋮
```

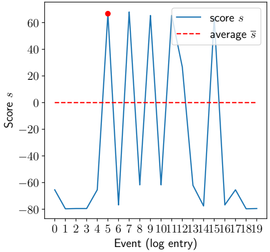


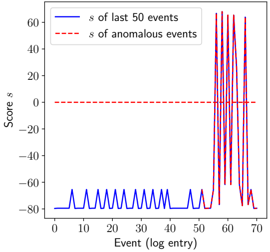


Example 2

⋮

0 remoteerrors errorcount $nz toggled $nz times $nz min
1 remoteerrors errorcount $nz
2 remoteerrors errorcount $nz
3 remoteerrors errorcount $nz
4 remoteerrors errorcount $nz toggled $nz times $nz min
5 **rpccheck clnt create error**
6 remoteerrors errorcount $nz
7 rpccheck fails $nz kill $nz
8 pid change $nz $nz
9 rpccheck fails $nz kill $nz
10 pid change $nz $nz
11 **rpccheck fails $nz kill $nz**
12 no process try start
13 pid change $nz $nz
14 remoteerrors errorcount $nz
15 no process try start toggled $nz times $nz min
16 remoteerrors errorcount $nz
17 remoteerrors errorcount $nz toggled $nz times $nz min
18 remoteerrors errorcount $nz
19 remoteerrors errorcount $nz
20 remoteerrors errorcount $nz toggled $nz times $nz min

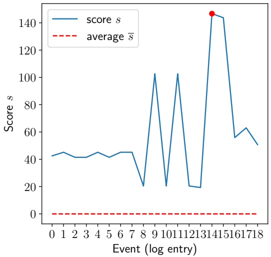


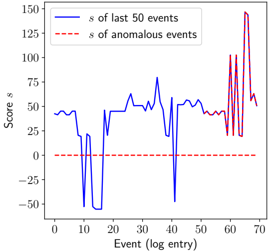


Example 3

⋮

0 getpid no process
1 no process try start
2 getpid no process
3 getpid no process
4 no process try start
5 getpid no process
6 no process try start
7 no process try start
8 pid change $nz $nz
9 getpid pid not match process name
10 pid change $nz $nz
11 getpid pid not match process name
12 pid change $nz $nz
13 pid change $nz $nz
14 pid not match process name toggled $nz times $nz min
15 pid not match process name toggled $nz times $nz min
16 signal term received
17 terminating threads closing files
18 writer thread terminated
19 interrupt thread terminated





Conclusion & Future Work

- The proposed method represents log entries as **word embeddings and models sequences** as HMMs to identify anomalies without labeled data.
- It detects **deviations from learned sequential patterns**.
- Results on logs from EuXFEL nodes show the approach can flag potential issues via score spikes corresponding to errors or disruptions.
- Challenges remain in **handling noise and minimizing false positives** in noisy logs.
- Future work could explore more advanced techniques and incorporate additional node statistics like **CPU/memory/network loads**
- The unsupervised sequence modeling approach enables detecting anomalies even when trained on logs containing anomalies, unlike supervised content-based methods. It focuses more on contextual irregularities than specific terms.

Thank you!

https://github.com/sulcantonin/LOG_ICALEPCS23

```python
from hmmlearn import hmm
import numpy as np

x = np.stack([[0,1],[1,0],[0,1],[1,0],[0,1],[1,0],[0,1],[1,0]])
model = hmm.GaussianHMM(n_components=2, covariance_type="diag")
model.fit(x[:-1,:])
logp = []
for i in range(1,x.shape[0]+1):
    logp.append(model.score(x[:i]))

logp = np.array(logp)
score = logp[:-1] - logp[1:]
```

Contact

Deutsches Elektronen-Synchrotron DESY

www.desy.de

Antonin Sulc, Annika Eichler, Tim Wilksen
0000-0001-7767-778X
MCS
antonin.sulc@desy.de