The Journal of Open Source Software

Unsupervised learning approach towards anomaly detection in compact logs with ADE

Ayush Shridhar[*1] **and James Caffrey**[2]

1 International Institute of Information Technology, Bhubaneswar, India **2** IBM, New York City, United States of America

DOI: 10.21105/joss.03052

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @arcuri82
- @mdpiper

Submitted: 19 January 2021
Published: 22 February 2021

Introduction

Anomaly detection engine (ADE) is a framework written in Java that detects anomalous time slices and messages in Linux and Spark logs using statistical learning. It does so by building a model of the expected behavior and then comparing it against the observed behavior for every time period. It then assigns a few statistically calculates scores: Bernoulli score, Poisson score and an anomaly score that indicates the chances of the corresponding time slice containing anomalous logs or being the cause of unexpected behavior of the system.

At the heart of ADE are unsupervised learning algorithms. Data pipelines extract important features from the logs messages that are then used to train different model groups. These trained models then contribute towards calculating the statistical scores. ADE outputs the score and all other relevant metadata as an XML file which can then be viewed in the browser. Not only does ADE report the calculated score for each time interval in the of the entire duration, but it also generates an internal summary for each interval. This includes the message ids that are generated using Levenshtein distance to group similar messages together, the corresponding messages, periodicity status, details about the occurrence of each message and their statistical scores.

Statement of Need

Anomaly detection is necessary to get to the root of the problem in case a system crashes or behaves unexpectedly. Such crashes can lead to loss of time and resources. To make things worse, getting to the bottom of the problem is a tough job: sysadmins typically needs to race against time and go through hours of log messages to identify the source of the issue. This task becomes more difficult when the density of the logs increases, which is very common in Spark or web server based applications. To make things worse, many applications involve a cluster of machines working in parallel. Unexpected behavior in a single machine can cause fatal crashes and inaccurate outputs from the cluster.

Here's where statistical learning steps in. ADE learns what ideal system looks like and then continuously goes through the generated log messages to produce a report summary for each interval of the observed data. The report summary contains various calculated metrics and an anomaly score for each interval. Greater anomaly score means that there's a higher chance of the corresponding interval containing anomalous logs, thus causing unwanted behavior. This makes it easier to realize the source of the problem, since we now know the time interval during which unexpected behavior was reported in the system. The task of going throught hours of logs messages has been narrowed down to analysis of analysing just the flagged intervals,

*Work done as Open Mainframe Project (The Linux Foundation) Mentorship program intern

since the model is sure that these have reported unusual messages. In additional, for each interval in the data, ADE produces a summary of each message seen during the interval. This directly points to the exact message that might be causing the issue.

There has been a lot of work towards the problem of anomaly detection. TadGAN (Geiger et al., 2020) is a generative approach towards solving this problem. Another way to solving this problem is by using a semi-supervised adversarial learning approach, as done by *GAN Augmented Text Anomaly Detection with Sequences of Deep Statistics* (Fadhel & Nyarko, 2019). Our approach is very different: we treat this as a statistical problem and avoid introducing computationally heavy deep learning into the picture. This makes the solution more explainable and faster to compute.

Novelties

ADE is written in the Java language that makes it ideal to be used across any platform without changing the underlying code.

ADE comes batteries-included: One can just build the binary and invoke required commands to run ADE on their data.

Minimal external dependencies: Most of the functionalities are directly implemented without replying on external libraries.

ADE does not require intensive data labelling Since ADE uses unsupervised learning algorithms, it does not require the data to be labelled. Any continious or streaming source of logs directly serves as the data required to train the model groups.

Anomaly detection engine (ADE)

Internals

ADE splits the incoming logs into time slices, called *analysis intervals* in ADE terminology. The length of these time slices can be varied depending on the application. Linux logs are generally less dense than Spark logs and thus have a larger interval size.

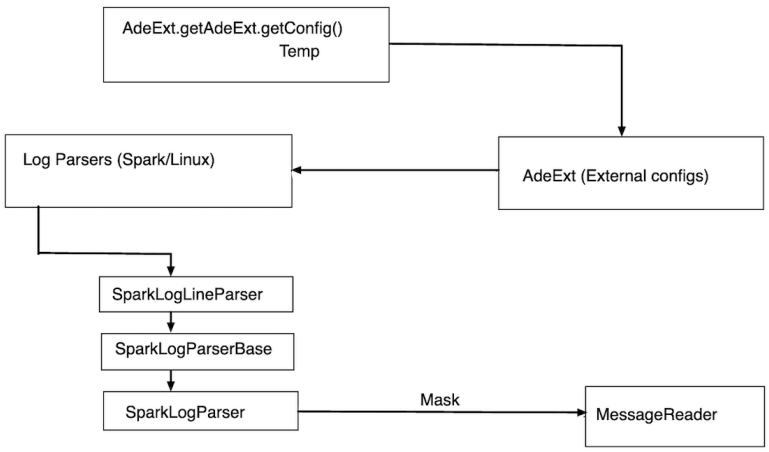
Internally, ADE processes these logs and assigns them message id, which is a unique identifier for the corresponding message. Message ids are assigned on the basis of Levenshtein distance. This allows us to assign logs with similar messages the same id, since they are probably going to exhibit similar behavior.The next step involves training the *model groups*, which are collections of one or more systems that handle similar type of workloads.

ADE takes into account four major features to measure how unusual an interval is:

- Number of unique message ids
- Interval anomaly score
- Number of messages not in the model
- Number of messages which have not been seen

ADE contains three major submodules: ***ade-core*** contains the core functionalities responsible for calculating scores, performing unsupervised clustering and modules to deal with input streams. The scores computed here include *Interval Anomaly score*, *Bernoulli Score*, *Best-of-two score*, *Log normal score*, *Poisson score*, *rarity* and *severity* scores. We also calculate the critical and unique words for each message that finally contributes to the score.

ade-ext is where the data stream parsing happens. It reads parameters specified in the setup file to invoke the corresponding parsers during runtime. Since each log message has a fixed format, the parsers use regular expression matching to extract the relevant fields. The logs are *masked* to remove unnecessary or sensitive data before sending them to the final *MessageReader* that sends them directly to the output stream.



Internal block diagram of ade-ext

ade-assembly contains scripts and metadata files (setup file, template XML files) to run the framework.

For each message, ADE tries to understand if the message was expected or not. This understanding (called the *context* of the message) forms a useful indication of the behavior of the system. ADE classifies every message into four possible *contexts*:

- New : Indicates messages that ADE has never seen before.
- IN_SYNC : Implies that ADE expects the message to be issued in a periodic pattern, and the message was issued as expected.
- NOT_IN_SYNC : Implies that ADE expects the message to be issued in a periodic pattern but the message was not expected.
- NOT_PERIODIC : Indicates that ADE does not expect the message to be periodic.

Usage

ADE uses maven package manager and Apache Derby database. Once the binary has been build, ADE provides various commands to invoke relevant functionalities.

- `controldb` Contains tools to performs actions on the database. This includes:
  - `create`: Create a database (with properties specified in `setup.props`)
  - `delete`: Delete contents of the database
  - `query "SQL statement"`: Perform the SQL query on the database
  - `dml "SQL statement"`: Issue the SQL statement
- `upload`: Uplooads the directory or compressed file to the database. Intenally, it carries out all the logs parsing and preprocessing operations.
- `verify`: Used to verify if the amount of data is sufficient to train the model groups.

- `train`: Trains the model groups. We can additionally specify the model groups to train, else it trains all model groups by default.
- `analyze`: Uses the train models to analyze previously unseen logs. Generates analysis files.

Output format

ADE outputs the analysis report in XML format. The report contains an analysis summary for the entire log duration, split on the basis of the interval size. Each interval contains an assigned anomaly score. For each interval, ADE generates and interval analysis file, also in XML format. This file gives finer intrinsic analysis information for the interval. This includes details about each message, periodicity status, frequency, Bernoulli score, Poisson score and anomaly score.

Interval Index	Interval Time	Anomaly Score	Number of Unique Messages	Num of New Msgs	Num of Never Seen Before Msgs	Missing	Reason for Missing	Interval, V2
0	0:00 - 0:10	99.5	32	7	0	false		XML
1	0:10 - 0:20	98.9	26	6	0	false		XML
2	0:20 - 0:30	98.9	28	5	0	false		XML
3	0:30 - 0:40	98.9	27	5	0	false		XML
4	0:40 - 0:50	99.5	30	8	0	false		XML
5	0:50 - 1:00	99.5	29	8	0	false		XML
6	1:00 - 1:10	100.0	28	10	0	false		XML
7	1:10 - 1:20	100.0	28	10	0	false		XML
8	1:20 - 1:30	100.0	25	10	0	false		XML
9	1:30 - 1:40	100.0	24	10	0	false		XML
10	1:40 - 1:50	99.5	21	7	0	false		XML
11	1:50 - 2:00	99.5	21	7	0	false		XML
12	2:00 - 2:10	99.5	19	6	0	false		XML
13	2:10 - 2:20	99.5	19	6	0	false		XML
14	2:20 - 2:30	99.5	19	6	0	false		XML
15	2:30 - 2:40	99.5	19	6	0	false		XML
16	2:40 - 2:50	99.5	19	6	0	false		XML
17	2:50 - 3:00	99.5	19	6	0	false		XML

Example of ADE analysis for a day. Note: Each period has an associated analysis XML file

Interval anomaly score: 99.5

Message Id	Time Line	cluster_context	Num of instance	Bernoulli score	Frequency	Periodicity status	Periodicity score	Last Seen	cont	Poisson score	Anomaly score	User Rules	Message
sshd_47		out_of_context	24	99.706	0.071_occ/day 0.994_occ/14days 340.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:05:30.000Z	6.921	1.0000	0.9990	N/A	Accepted keyboard-interactive/pam for user1 from 5dbf2b6d08477c5fc7774f590b31abd4f39be38a port * ssh2
/usr/sbin/cron(notes): /home/notes /getsysdata.ksh_311		new	4	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:00:00.000Z	4.656	0.0000	0.9905	N/A	(notes) CMD (/home/notes/getsysdata.ksh 1>/home/notes/run.out 2>&1)
/usr/sbin /cron(notesa): /home/notes /getdomdata.ksh_309		new	4	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:00:00.000Z	4.656	0.0000	0.9905	N/A	(notesa) CMD (/home/notes/getdomdata.ksh 1>~/doms.out 2>&1)
/usr/sbin /cron(notesb): /home/notes /getdomdata.ksh_312		new	4	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:00:00.000Z	4.656	0.0000	0.9905	N/A	(notesb) CMD (/home/notes/getdomdata.ksh 1>~/doms.out 2>&1)
sshd_281		new	25	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:07:00.000Z	4.656	0.0000	0.9905	N/A	subsystem request for sftp by user *
sshd_295		new	2	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:09:00.000Z	4.656	0.0000	0.9905	N/A	error: Received disconnect from 7300101c7c32b48044f1dd27842d6474546d7cf7: 10: General disconnection
sudo(itimssh):/usr /sbin/faillog_294		new	1	101.000	0.071_occ/day 0.994_occ/14days 338.000_interval/occ	NOT_PERIODIC	NaN	2015-12-11T23:07:00.000Z	4.656	0.0000	0.9905	N/A	itimssh : TTY=unknown ; PWD=/home/itimssh ; USER=root ; COMMAND=/usr/sbin/faillog -u ytjvijay -r

Example of ADE analysis for a period

ADE for Spark

Running ADE for Linux or Spark can be toggled externally via the setup file. This stores all the metadata required to run the commands. If `ade.useSparkLogs=True`, ADE internally invokes all Spark log parsing and processing functionalities, else it falls back to Linux Syslogs.

Acknowledgements

We'd like to thank the Open Mainframe Project, Linux Foundation for funding the expansion of this project to include middleware logs as a part of their internship program. We'd also like to thank the maintainers and developers of *LogHub* (He et al., 2020) for providing us high quality data to train and test the system.

ADE has been supported by the Open Mainframe Project.

References

Fadhel, M. B., & Nyarko, K. (2019). GAN augmented text anomaly detection with sequences of deep statistics. *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. https://doi.org/10.1109/ciss.2019.8693024

Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A., & Veeramachaneni, K. (2020). *TadGAN: Time series anomaly detection using generative adversarial networks*. http://arxiv.org/abs/2009.07769

He, S., Zhu, J., He, P., & Lyu, M. R. (2020). *Loghub: A large collection of system log datasets towards automated log analytics*. http://arxiv.org/abs/2008.06448