Additional log support for Anomaly Detection Engine

Ayush Shridhar

Contents

1 Abstract

This project aims at upgrading and enhancing *OpenMainframeProject/ade*, by implementing parsers for Spark logs. The ADE project was started in early 2016, with the purpose of making prevalent Linux systems more stable by implementing a framework to help in quick traceablility of the source of an unusual occurrence by predicting anomalous logs from Linux Syslogs. Analysis of the anomalous logs can be a good indication of the root of the problem, thus helping in eradicating such issues. This is specially important when working in an environment where every second is important, and fatal system crashes can mean the loss of tens of thousands of dollars. Such systems, for example stock markets, can be hugely benefited by a system that identifies the origin of the anomaly, which can be the cause of a fatal crash.

However, ADE supports only Linux Syslogs at the moment. With this project, I aim to enhance ADE by expanding it to support middleware areas, such as nginx/spark logs. I hope this project is a solid move towards more robust anomaly detection. It should also help in showcasing the ability of ADE to deliver results in the case of comparatively more complicated logs.

2 Background on ADE

2.1 What is ADE?

The ADE project was created to make Linux systems more stable by helping in identifying the reasons behind system crashes. ADE detects anomalous logs from a group of logs. These anomalous logs can give a good indication of what might have caused unwanted events to occur. Studying these logs can help in narrowing the cause of the problem to a few particular issues, thus identifying the root of the problem.

ADE takes the data science approach to solve this issue. It uses machine learning algorithms that are trained on huge amounts of data to predict if an incoming log slice obeys various conditions that the classifier implicitly learns from the data.

2.2 Why do we need ADE?

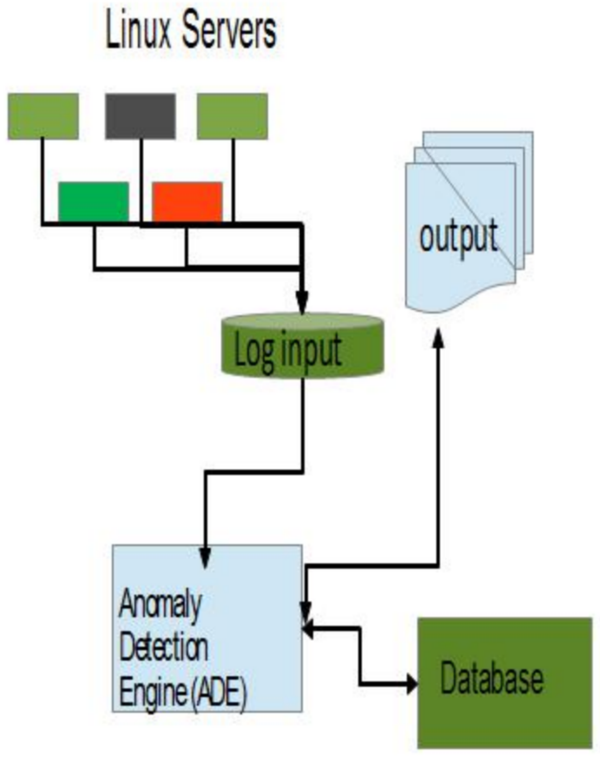
Linux is the most widely used operating system today. Embedded Systems, Supercomputers, Web Servers and mobile devices, all run Linux. Naturally, there is a need for a system that can help in detecting the root of an unwanted occurrence in the system. This is especially important in the case of stock trading firms, that depend of the stability and performance of systems for trading. Each second matters in this line of work. According to an Economic Times study, trading is halted many times each day due to faulty systems. It is essential to identify the source of the error and fix it to avoid occurrence of such phenomena again. That's where ADE comes in. It narrows down the search to a few logs that differ from most of the other logs. An analysis of these logs can lead to identifying the root of the issue. Other than this, we might need more fine grained information about an event:

- When did this activity start?
- Were there more than usual messages during this interval?
- Were there new, previously unseen messages during this period?
- Were there more occurrences of these messages?

and so on.

2.3 How does ADE work?

At the core of ADE is a set of unsupervised machine learning algorithms. Such algorithms, also known as clustering algorithms, can identify logs that vastly differ from most of the other logs.



The ADE analysis results are written to files in XML format, which can be viewed using a web browser or used in other processing to support the enterprise Linux IT solution that is generating the logs.

For each time slice (interval), ADE measures how unusual the interval is by:

- Calculating an anomaly score that describes how unusual the time slice is
- Determining the number of similar message strings within an interval
- Determining the number of new message strings within an interval

ADE creates a summary file with this information for all of the time slices (interval) within a day (period). Here is an example of the summary file for one day.

3 Improvements proposed

3.1 Overview of improvements proposed

The overall aim of this project is to extend ADE to accommodate Spark logs. This comes with a set of complications, that I'll discuss later. ADE has shown to be constructive while dealing with Linux Syslogs. Testing its performance on Spark logs, that are generally more complicated than Syslogs, can showcase the true potential of ADE and expand its usefulness to middleware areas. The various steps in the process of accommodating these logs are:

- Data pre-processing and sanitizing
- Implementing of a parser to process the logs
- If needed, implementing a data pipeline for input into the classifier
- Experimenting with the model and training the classifier
- Analysing the results of the model
- Investigating the reason behind unusual results (if any), and working towards fixing these.

And subsequently building up on top of this.

3.2 Dataset to be used

I'm grateful to the researchers from the Chinese University of Hong Kong, who've generously agreed to grant me access to the data I'd need for the project. The dataset is about 2.75 GBs large, and contains about 33 million log entries. The log set was collected by aggregating logs from the Spark system in a lab at CUHK, which comprises a total of 32 machines. The logs were aggregated at the machine level and provided as-is without further modification or labelling, which involve both normal and abnormal application runs.
More information about the dataset can be found on their website:
http://www.logpai.com/

3.3 Spark logs

Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. On top of the Spark core data processing engine, there are libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application. Programming languages supported by Spark include: Java, Python, Scala, and R. Application developers and data scientists incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, IoT, or financial systems, and machine learning tasks.
Logging is naturally an important part of this process. Logs are not only for debugging and traceability, but also for business intelligence. Building a robust logging system within applications could be used to obtain insights of the business problems being solved. Spark uses log4j as the standard library for its own logging. Everything that happens inside Spark gets logged to the shell console and to the configured underlying storage.

3.4 Details of the improvements

3.4.1 Data pre-processing and sanitizing

An important part of any machine learning task is ensuring that the data is good enough to be used. The dataset I plan to work with is dirty; it has been provided as it was generated, without any modifications. Naturally, it is important to transform this raw data into an understandable format. This is more important while dealing with text data. Cleaning and exploring the data forms an essential base for future steps.

3.4.2 Implementing a parser to process the logs

Once we have the cleaned data, we need to pass it through a parser to bring it into a format that can be feeded into the machine learning classifier.

3.4.3 Data pipeline

In various cases, a data pipeline can be helpful. Such a pipeline consists of a number of transformations. Each transformation amends the data. The end result of this pipeline would be the data in the exact format needed as input into the classifier.

3.4.4 Training & experimenting with the classifier

Once we have the data in required format, the next step is to train the machine learning classifier on the data. This involves experimenting with hyperparameters (externally controlled parameters) of the model, to obtain the best results.

3.4.5 Analysis of results & investigating

This involves studying the results of the classifier, and comparing with ground truth. While working with Data Science tech stack, feature engineering and dimensionality reduction are techniques used to ensure the data features aren't correlated, which might cause poor performance of the model.

4 Timeline

Week	**Deliverables**
Community bonding period	• Study, build and play around with the code to get a feel of the existing structure of the project. • Get an idea of the modules that would need to be amended. • Get all doubts clarified before moving to the coding phase.
June 1 - June 15	• Study and analyse the dataset • Implement modules for data pre-processing and write tests
June 16 - June 30	• Start working on a parser to explicate the processed data. • Test and verify the functioning of the parser on real world data.
July 1 - July 8	• Buffer week: Ensure no work from previous weeks is left and fix bugs that arise in the process.

Week	**Deliverables**
July 9 - July 23	• Study the relevance and importance of a seperate data pipeline in this case, and if it could help solve issues with pre-processing + parser.
July 24 - August 7	• Train and experiment with the classifier. • Analyze and compare the results of the classifier.
August 8 - August 15	• Buffer week: Ensure no work from previous weeks is left and fix bugs that arise in the process.
August 16 - August 30	• Investigate results behind unwanted results and work on fixing those. This might involve re-training the model with difference parameters and implementing additional pre-processing steps.
August 31 - September 14	• Complete all work I might have left for later. • Ensure all objectives have been met and test pass. • Submit code, tests and write documentation.

5 About me

My name is Ayush Shridhar. I'm a final year undergraduate, majoring in Computer Science. I'm fascinated by Computers and potential of technology to touch different aspects of our life. I started studying about machine learning during my junior year, and I've been hooked since. I love exploring different technologies and solving problems using these. I'm an open source aficionado, with contributions to Mozilla, TensorFlow, JuliaLang among others. Apart from computers, I love watching football and listening to music. I'm pretty active online and you can contact me on:

- Mail: ayush.shridhar1999@gmail.com
- OpenMainframeProject slack: Ayush
- Github: @ayush1999 / @ayush-1506

6 Why me?

I love exploring and solving problems. I'm extremely grateful to the open source community for guiding me during my initial years in this field. This helped me shape my thoughts and realise my interest in software development and machine learning algorithms. I also got to interact with amazing people in this process. I love reading about recent advances in AI and implementing these myself. I love working on side projects that solve different issues. I'm always excited about working on projects that can affect others in a positive way. My previous experiences include:

- Google Summer of Code student with The Julia Language, NumFOCUS in 2018: Developing a parser for ONNX serialized deep learning models for the Flux.jl framework,
- Visiting Researcher, University College London: worked under Prof. Franz Kiraly in the Department of Statistical Science on implementing a neural network interface for the MLJ machine learning toolbox.
- Google Summer of Code student with Mozilla in 2019: Contributing to bugbug, a framework that applies machine learning algorithms on software engineering, thus automating issue tracking.
- Visiting Researcher, Australian National University: Working in the Robust Decision-making and Learning Lab on implementing reinforcement learning algorithms to solve dynamic target tracking problems.

By collaborating with and learning from experienced programmers from the industry, I hope to gain more experience, increase my skill set and become a better programmer, student and person in general. This would give me a much needed boost towards a career in the industry.