Analyzing raw log files to find execution anomalies

A novel approach to analyzing text-based log files to find changes in the execution profile of complex IT systems

Viktor Jovanoski
Carvic d.o.o.
Kotnikova 5
Ljubljana, Slovenia
viktor@carvic.si

Jan Rupnik
Jozef Stefan Institute
Jamova 39
Ljubljana, Slovenia
jan.rupnik@ijs.si

Mario Karlovčec
Jozef Stefan Institute
Jamova 39
Ljubljana, Slovenia
mario.karlovcec@ijs.si

Blaž Fortuna
Jozef Stefan Institute
Jamova 39
Ljubljana, Slovenia
blaz.fortuna@ijs.si

ABSTRACT

Anomaly detection (a.k.a. outlier detection) is the identification of events that do not conform to an expected pattern in a dataset. When applied to monitoring modern, complex IT systems, it keeps track of a plethora of incoming data streams. This paper provides an approach that uses the lowest and most unstructured source of data related to an IT system - the raw system log files. Several versions and parametrizations of basic building blocks will be presented to show how different types of anomalies can be extracted from the data. Several experiments on synthetic as well as real-world data show effectiveness of the algorithm. Special care is taken to keep the model and the resulting alerts interpret-able - since detecting an error without a meaningful explanation about its details is of limited use to end user (the results need to be actionable).

Keywords

Anomaly detection, Outlier detection, Infrastructure monitoring

1. INTRODUCTION

Modern IT systems are getting increasingly complex and distributed. On-line monitoring of their health is becoming critical for their normal operation. The data that is being collected about these systems comes in diverse time series (numerical, categorical, text), potentially huge in volume and arriving with different latencies. For instance, complex systems often expose metrics about their performance such as number of served requests or size of internal data structures. One can also monitor systems performance indirectly by inspecting CPU load, network load or database communication patterns. Quality assurance of input and output data can also be performed, such as the number and the size of the data records. Spotting unusual behavior of such systems is crucial and unhandled execution problems can lead to catastrophic results. The anomalies can be very different in nature, from abrupt changes that occur within a second to the gradual decay of performance that is only observable on a weekly or monthly scale.

Most of the research about anomaly detection has been concentrated on outlier detection in numerical time-series (e.g. financial time series like in [5], time series arising from infrastructure monitoring) and discrete-event sequences (e.g. fraud detection like in [4], cyber-security applications like in [3]). Data representation (the way of encoding the relevant information) is vital to the practical performance of an anomaly detection approach (does it capture relevant events or just insignificant variation).

However, all of these numeric data series have to be "prepared" in advance. A developer had to think ahead about the potential problems that can arise and expose appropriate measurements. When this data is available, it can clearly signal the problem to the operators. But what happens when a new type of outage occurs and no measure to detect it was put in place? In such case the infrastructure maintainers typically resort to inspection of *raw log files*. Writing a line of text to console or file output is often the only indication that something happened or has not happened when it should have. In our experience, all complex IT systems produce such files and they are still used to solve the hardest problems and errors.

The log files may be very unstructured in practice and may contain extremely diverse information. From errors, warnings, database calls and initialization steps, to casual counters and observations. The log lines themselves may be unstructured: their content might be unordered and they may contain text written in natural language (e.g. error messages). Even then, these bits of information may or may not be written in a easy-to-parse form (e.g. JSON format). The only thing that can be expected of each line is a *timestamp* - a clear indication of time on the server, when this particular line was written to the log. Even if this data is missing from some lines, the sequential order of writing to file helps us narrow down the possible timestamp for each line. We can simply choose to re-use the last timestamp before that line.

Many algorithms and approaches to anomaly detection have been proposed in the literature. [1] provides an excellent overview of the field. New approaches are getting increasingly more sophisticated at dealing with multidimensional numeric data, discrete, sequential or even spatial data. The unstructured nature of raw log files makes the problem amendable to text-mining based approaches. This article presents work in that direction.

2. ANOMALY-DETECTION

End-users in charge of maintaining a large IT system will typically be concerned with two types of anomalies. Either they will want to avoid a sudden, critical degradation of performance, or they will want to know if the performance has been slowly degrading and attempt to prevent that.

Abrupt change - In this scenario, the system "falls of the cliff" - performance plummets and multiple parts of the system typically experience severe problems. Users will most often be notified of the problem via many channels. Hence, the raw log files are not the first place they will start looking at. However, if the cause of the problem cannot be determined or the exact timeline of events of the disaster cannot be established, the information from the log files will also be used as a part of the forensic analysis.

Gradual deterioration - IT systems that have been in production environment for a long time can experience gradual changes. These changes do not necessarily cause catastrophic failure overnight, but degrade the performance over a longer period of time. Such changes are very difficult to detect by the programmer as they are very subtle, e.g. they are only observable on the monthly scale. And often the only place where this can be detected is with a long term analysis based on the raw log files.

2.1 General pipeline

Algorithm 1 shows the general structure of a typical anomaly-detection system that operates on a stream.

First, a data record is extracted from the incoming data. As stated before, the timestamp is always defined. The record can also incorporate recent past data (previous lines of log file) and contextual data (e.g. server overall status, holidays indicator).

In the second step a *score* is calculated that should reflect a record's anomalousness (non-normality, novelty). The simpler the score, the easier the subsequent steps are. Additionally, we prefer models and scoring functions that can easily be explained to the end user, since he is supposed to act on them. Lastly, the score should be constructed in such manner that the anomalous examples fall on one edge of the spectrum (e.g. the higher the score, the bigger the anomaly).

Algorithm 1 General anomaly pipeline

```
while input data available do
    parse data and extract record
    calculate anomaly score
    if score above threshold then
        report anomaly
    end if
    add record to model
end while
```

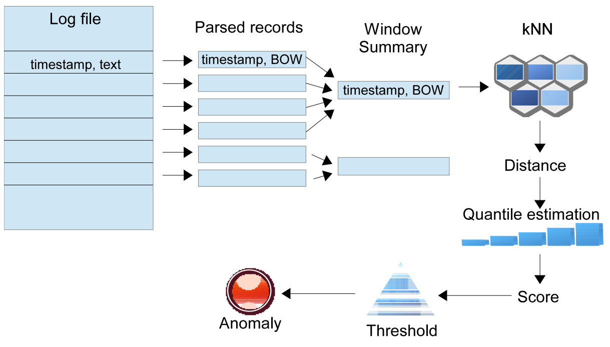

Figure 1: Text-processing anomaly pipeline

The score value is then used to decide if the incoming data record is an anomaly - either by comparing against some manually predefined static threshold or against a dynamic one, which uses historical score values to determine the threshold autonomously. Finally, in case of an anomaly, an alert is created that contains enough data to explain what was observed and why it was tagged as an anomaly.

Only when all of the above steps are done, we add the new data point to the model. We can store it internally in the model for some time, but we have no guarantee to be able to ever again access all the historical data (e.g. for retraining of the algorithm).

2.2 Defining normality

It is crucial for any anomaly-detection system to be able to tell what *normal* is, so that based on this notion it can say that something is *not normal*, i.e. anomalous. When available, domain knowledge and experience from human experts can be used to capture the appropriate aspect of normality in the data, which greatly improves the quality of the reported anomalies.

In general, however, the systems need the ability to autonomously define normality. This ability greatly depends on the scoring function and it is common practice to define it in a way that only the values on the border of observed values are indicators of an anomaly.

2.3 Detecting anomalies on raw log files

We will now present our approach to anomaly detection based on text processing techniques. The steps of the approach are given in Algorithm 2 and graphically presented in Figure 1.

The log parser processes a single line at the time and each

Algorithm 2 Log-file anomaly pipeline

```
while input data available do
    read data from log file
    parse data and extract BOW
    insert into time windows
    calculate distance in kNN
    if score quantile above threshold then
        create explanation using distance
        report anomaly
    end if
    add record to model
end while
```

time emits a record with several features. In the second step we extract standard *BOW* (bag-of-words) feature vectors. There are several possible ways of how to extract features and how to weight each feature dimension. We chose to use a simple representation where we extract tags, such as *server=x* and *process=mytask.tsk*. We collect all these tags from the record and assign them weight 1. This means that individual lines can produce vectors of varying lengths, as they are not normalized. We could also normalize them or re-weight them using the TFIDF weighting scheme ([6]) to down-weight frequent tags.

To capture the wider context of each record, we aggregate a set of records within a time-window of predefined length and emit a combined record (simple normalized sum of all vectors) that represents that window. In the fourth step, we use the *k-nearest-neighbor* algorithm (kNN) to find which k windows from the past are the closest to current window. This unsupervised algorithm was chosen because it can handle skewed distributions very well.

The average distance of the k neighbors is used as an *anomaly score*: the further away an instance is to its nearest neighbor set, the more anomalous it is. The value of the parameter k is usually small, between 1 and 5. The best value depends on the data domain and should be determined by experimenting.

The absolute scale of the score may vary from problem to problem and also depends on the feature representation. For that reason we used a quantile based approach: we compare the anomaly score of a new instance with scores of recently observed data points (the size of the of recently observed data point set is controlled with a parameter *learning window*). If the score is higher than a large (example 0.999) fraction of scores (controlled by a parameter *nn_rate*), then we classify the instance as an anomaly. The quantile (1 - *nn_rate*) directly controls the detection rate (0.001 corresponds to classifying 0.1% of instances as anomalies) under the assumption that the data distribution is stationary.

2.3.1 Tokenization

When tokenizing input tests we have several options. If we know that some common patterns exist on how the special entities are marked inside the text, we can extract them and create useful features for *BOW* step. For instance we can use message text directly to create BOW. Alternatively, we can just find identifiers of the origin of the message (e.g. process name, method name, class name, page name etc.) and use these instead of whole texts.

Table 1: Synthetic Data - window = 1 min

NN rate	Precision	Recall
0.003	0.06	1.00
0.001	0.15	0.66
0.0005	0.23	0.66

Table 2: Synthetic Data - window = 1 h

NN rate	Precision	Recall
0.05	0.07	0.66
0.03	0.14	0.66
0.01	0.33	0.33

2.3.2 Anomaly explanation

We construct the explanation for an individual alert by finding its nearest neighbor and subtracting one vector from the other and squaring each element of the resulting vector. Each dimension is thus attributed with an *"anomalousness"* score, and the highest scoring dimensions contributed the most to the distance to the nearest neighbor. The explanation to the user contains a sorted list of highest scoring dimensions (clipped to avoid information overload).

3. EXPERIMENTAL RESULTS

3.1 Evaluation

In most real-world anomaly-detection cases we receive a dataset that is largely unlabelled. The labels we have usually denote some special catastrophic situations that users experienced and want to avoid in the future. The rest of the data can be assumed to be mostly *"normal"*, but unknown anomalies may also remain in the dataset. The standard metrics to evaluate the performance of an anomaly detector are *precision* (#true anomalies / #predicted anomalies) and *recall* (#predicted true anomalies / #true anomalies). Low precision translates to a higher burden on the user that inspects the anomalies (each inspection has some cost) and low recall translates to missing many anomalies and increasing risk. If we suspect that the dataset is not labelled completely (unlabelled anomalies present) the precision might be estimated pessimistically and recall might be measured optimistically. In such cases manual inspection of false positives might lead to discovering new relevant types of anomalies present in the dataset.

3.2 Synthetic data

We generated log files by simulating parallel execution of several processes, each having a specific pattern of writing to log. We then manually inject 8 instances of anomalous entries, 2 per week, each pair occurring within one minute.

We use 10 days for the kNN learning window length, so there will be no anomalies in the first 10 days. For the length of the input-grouping window, we experimented with two settings: 1 minute and 1 hour. In the former setting we have 6 original anomalies, but in the later, we only have 3 since each pair of adjacent anomalies (they are 1 minute apart) gets collapsed into the same hour. We set the parameter k to 1 - thus making the explanation of the anomaly very simple.

Table 3: Web logs - an anomaly explanation example

File	Val	Near	Contr
/shuttle/missions/sts-68/mission-sts-68.html	0.707	0	0.354
/images/NASA-logosmall.gif	0	0.505	0.180
/htbin/cdt_main.pl	0	0.416	0.122

Table 1 shows the results for windows of length 1 minute. When parameter *nn_rate*, which controls the sensitivity to outliers, was set to 0.03, it correctly detected all 6 manually inserted anomalies. Table 2 shows the results for windows of length 1 hour. This granularity of data is too coarse and hides certain anomalies that remain undetected.

3.3 Web-server logs

We analyzed browsing pattern logs from a production web server where the logs contained information on web-page and file requests. The specific feature of this web site is that it is almost completely static - there are almost no new pages being added to it, so the browsing patterns should be relatively constant. The dataset spanned a period of one month. We set the summarization window length to 5 minutes and the kNN comparison window length to 10 days. The parameter k was again set to 1 and the anomaly rate was set to 0.001. The parameters were hand-tuned.

When analyzing such data the anomalies that we are interested should capture both system failure (malfunctioning software, infrastructure failure) as well as malevolent behavior (denial-of-service attack (DoS), a hacker-induced scanning for exploits). In these cases the system should ideally produce anomalies with **strong dimensional outliers**.

We manually inspected anomalies where the strongest dimension contributed more than 30% of the total anomaly score. An example is shown in Table 3, where columns Val means value in current record, $Near$ means value in the nearest record and $Contr$ means contribution to the total distance. It turns out that these anomalies corresponded to the rare events when new content was added to the web page. No DoS or hacker attacks were detected in the observed time period. So the vector dimension for the file ($mission - sts - 68.htm$) was $Val = 0.707$. This file was not present in the nearest record ($Near = 0$) and this dimension contributed 0.354 of the total distance between this record and the nearest one.

4. CONCLUSIONS AND FUTURE WORK

We presented a novel combination of known approaches to anomaly detection using techniques developed in the field of text mining. Using these we were able to extract valuable information from raw textual log files that are normally only used for manual inspection and forensic analysis.

Our algorithm is based on processing log files, but many other sources of information can be used to extract anomalies in modern IT systems. Our long-term goal is to design what we call *Full-spectrum anomaly detection system* (FSADS) that will be able to import many different types of data streams, covering a wide range of aspects of an IT system (inputs, outputs, internal performance, database performance, network communications etc.). After each single source of data is analyzed and anomalies are extracted, the next step in FSADS will correlate them, determine critical signals (which anomalies have a high impact on system?), indicate possible root causes (what might have caused a particular anomaly?) and give predictions (what may follow after detecting a particular type of anomaly?).

The popularity of deep learning techniques ([7]) is also felt in the anomaly-detection field and we plan to study their application to multivariate analysis. Most often, autoencoders are used to create compact descriptions of the data and may also be used to highlight the dimensions with high reconstruction error. Another interesting approach is to use generative adversarial networks ([2]), where two neural networks are contesting in a zero-sum game: one network generates "normal" candidates and the other one discriminates between generated examples and true examples. We currently see two major challenges for broader use of deep neural networks in anomaly detection systems. The first one is the ability to **explain the results** to the end-user in an actionable way. The second one is processing of **stream data** and updating the model on-the-fly. Currently, existing and simpler techniques (e.g. kNN, clustering, statistical tests) provide much better support for these requirements. However, the linearity in describing the feature space is an issue that we would like to address in the future and deep neural networks present a promising approach.

5. REFERENCES

[1] C. C. Aggarwal. *Outlier Analysis.* Springer New York, New York, New York, 2013.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[3] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 219–230, New York, NY, USA, 2004. ACM.

[4] X. Liu, P. Zhang, and D. Zeng. Sequence matching for suspicious activity detection in anti-money laundering. In C. C. Yang, H. Chen, M. Chau, K. Chang, S.-D. Lang, P. S. Chen, R. Hsieh, D. Zeng, F.-Y. Wang, K. M. Carley, W. Mao, and J. Zhan, editors, *ISI Workshops*, volume 5075 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2008.

[5] C. Phua, V. C. S. Lee, K. Smith-Miles, and R. W. Gayler. A comprehensive survey of data mining-based fraud detection research. *CoRR*, abs/1009.6119, 2010.

[6] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975.

[7] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang. Deep Structured Energy Based Models for Anomaly Detection. *ArXiv e-prints*, May 2016.