FSADA, an anomaly detection approach

A modern, cloud-based approach to anomaly-detection, capable of monitoring complex IT systems

Viktor Jovanoski
Jozef Stefan International Postgraduate School
Jamova 39
Ljubljana, Slovenia
viktor@carvic.si

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

ABSTRACT

Modern IT systems are becoming increasingly complex and inter-connected, spanning over a range of computing devices. As software systems are being split into modules and services, coupled with an increasing parallelization, detecting and managing anomalies in such environments is hard. In practice, certain localized areas and subsystems provide strong monitoring support, but cross-system error-correlation, root-cause analysis and prediction are an elusive target.

We propose a general approach to what we call *Full-spectrum anomaly detection* - an architecture that is able to detect local anomalies on data from various sources as well as creating high-level alerts utilizing background knowledge, historical data and forecast models. The methodology can be implemented either completely or partially.

Keywords

Anomaly detection, Outlier detection, Infrastructure monitoring, Cloud

1. INTRODUCTION

Modern IT systems need several key capabilities, apart from tracking and directing the underlying businesses. They need to manage errors and failures - predict them in advance, detect them in their early stages, help limit the scope of the damage and mitigate their consequences. All this is achieved by analyzing past and current data and detecting outliers in it.

Anomaly detection must happen in near-real time, while simultaneously analyzing potentially thousands of data points per second. Incoming data that such a system can monitor is very diverse. Data can come in different shapes (numeric, discrete or text), in regular time intervals or sporadically, in huge volumes or just a few data points per day. Designing a system that can cope with such diverse situations can be challenging.

Another important aspect is *"actionability"* of the reported anomalies. When human operator is presented with a new alert, the message as to what is wrong needs to be clear. The operator must be able to immediately start addressing the problem. Sometimes all we need is a different presentation of the result, but most often the easy-to-describe algorithms and models are used - e.g. linear regression or nearest neighbour.

This high velocity of data (volume and rate) makes some of the algorithms less usable in such scenarios - specifically batch processing that requires random access to all past data is not desired. Ideally, we would only use *streaming algorithms* - algorithms that live on the stream of incoming data, where each data point is processed only once and then discarded.

The contribution of this paper is a hollistic approach to anomaly detection system that clearly defines different parts and stages of the processing, including active learning as a crucial part of the processing loop. The design addresses modern challenges in IT system monitoring and is suitable for cloud deployment.

2. ANOMALY-DETECTION

The most common definition of an anomaly is *a data point that is significantly different from the majority of other data points.* See [2] for a detailed explanation. This definition is strictly analytical. But most often the users define it within the scope of their operation, such as finding abnormal engine performance in order to prevent catastrofic failure, flagging unexpected delays in manufacturing pipeline in order to prevent shipment bottlenecks, detecting unusual user behavior that indicates intrusion and identifying market sectors that exibit unusual trends to detect fraud and tax evasion.

The anomaly-detection process is thus heavily influenced by the target domain. It also needs process-specific way of measuring the detection efficiency. For instance, in classification problems we can use several established measures such as *accuracy*, *recall*, *precision* or $F1$. In anomaly detection, on the other hand, we often don't have classes to work with

and secondly, we need strong user feedback to evaluate our results. Sometimes anomaly detection looks more like a forecasting and optimization problem. We measure how much the current state of a complex system is different from the optimal or predicted value.

2.1 Actionability

It is not sufficient for algorithms to just detect unusual patterns. Human operators that get notified about them must clearly understand the detected problems and be able to act upon them - we call this property of alerts *actionability*. For instance, it is not enough to report "the euclidian distance between multi-dimensional vectors of regularized input values is too big" - end-users will have no clue about what is wrong here. Instead, the system should report something like "The average processing time of customer orders is well above its usual values. This situation will very likely result in a significant drop of daily productivity." Some algorithms produce models that are easier to translate into human language than others. This feature needs to be taken into account when an anomaly-detection system is being implemented.

2.2 Modern challenges

In the era of big data there are many systems that produce data and a lot of the generated data can be used to monitor, maintain and improve the target system. The data volumes are staggering and need to be addressed properly within the system implementation.

Users expect results to be available as soon as possible - within hours, sometimes even minutes or seconds. In cases where automated response in possible, this time-frame shortens to miliseconds (e.g. stock trading, network intrusion).

Current systems for anomaly detection are developed as addons to the existing systems for collecting and processing data. This makes sense, since they developed organically, during the usage by the competent users, which identified areas that require advanced monitoring. We belive this provides necessary business validation of anomaly detection systems. However, there are limitations of such approach.

- Data that is available in one part of the system might **not be available** in another part, where anomaly-detection could greatly benefit from it.
- **Data volume** could prove to be too big for effective anomaly detection analysis, because needed resources might not be available (e.g. computing power is needed for main processing and anomaly detection should not interfere with it).
- Anomaly detection has **local scope** as it only processes data from one part of the system. The alerts are thus not aware of the potential problems in other parts of the system, so resolving issues takes longer and involves more people from several departments to coordinate during problem escalations.
- There is no systematic way of collecting the **user feedback** that would guide and improve the anomaly detection process.

3. THE SYSTEM ARCHITECTURE

To create a system that is able to ingest such huge amount of different data streams, detect anomalies in them and present user with a manageable amount of actionable alerts we propose a reference architecture of such system (figure 1). The acronym **FSADA** stands for *Full-Spectrum Anomaly Detection Architecture*, is based on the Kappa architecture [5] and comprises the components described below.

- **Storage module** contains historical data (raw and derived), background knowledge as well as generated alerts and incidents.
- **Stream-processing module** performs incoming-data pre-processing, as well as signal- and incident-detection.
- **Batch processing module** calculates aggregations, pattern discovery as well as background knowledge refresh.
- **User-interface module** (commonly abbreviated as GUI) displays raw-data, generated alerts along with feedback and active learning support.

3.1 Terminology

From now on we will be using the following terminology:

anomalies - any kind of abnormal behavior inside the system, regardless of the effect on the system performance.

signals - low-level anomalies that have been detected on single data-stream.

incidents - complex anomaly or a group of them with major impact on the system. Its time duration is usually limited to several minutes or hours. They are closely related to the way users perceive the system problems and outages.

alerts - an anomaly that is reported to the user, self-contained with explanation and basic context.

3.2 Storage module

The system needs to store several types of data that perform different functions. Each part of the storage layer can be located in separate system that best matches the requirements.

Measurement data represents raw values that were observed and processed in order to monitor the system. This data is strictly speaking not necessary when our algorithms are designed to work on a stream, but they are required for batch algorithms, for model retraining, active learning and for ad-hoc analytics. **Generated signals and incidents** are stored, additionally processed and viewed by the user. The storage needs to support flexible format of alerts, since each one of them is ideally an independent chunk of data that can be visualized without additional data retrieval. The algorithms can use **domain knowledge** to guide their execution. To facilitate this, the data needs to be stored in a storage system that provides fast searching, in order to be used in stream processing steps for enrichment, routing and aggregation. The algorithms inside the system create and

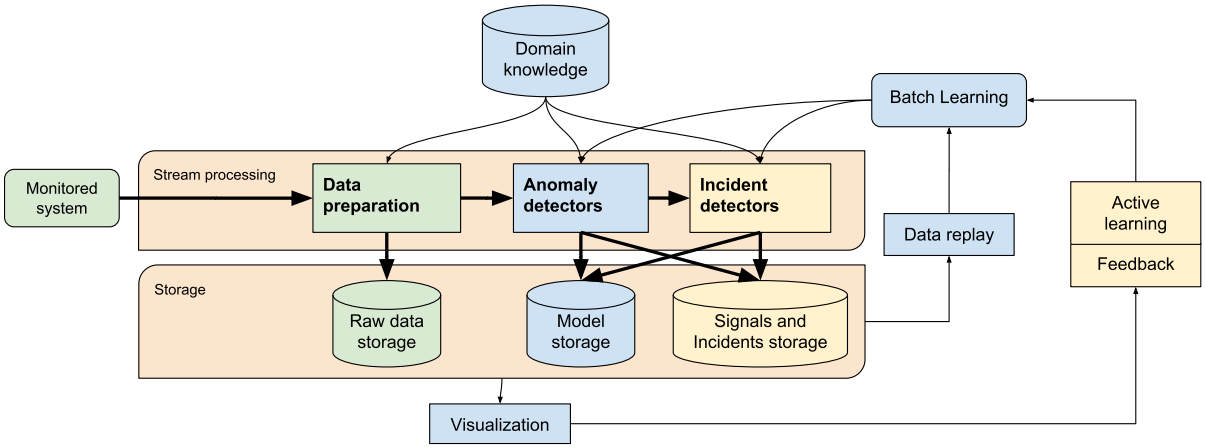

Figure 1: The big picture - displays the main building layers such as stream processing and storage, as well as the flow of the data between different components of the system.

update their **models** all the time, so this part of the storage needs to support reliable and robust storing of possibly large binary files.

3.3 Stream processing module

This module contains the most important part of the system - the components that transport the data, run the processing and generate alerts.

3.3.1 Incoming data pre-processing

Incoming data that the system analyses arrives at different volumes and speeds (*high-velocity*), as well as in many different types and formats. This data needs to be pre-processed before it reaches any anomaly detection algorithms.

Coping with such high-volume data stream requires special technologies. Streaming solutions such as Apache Kafka [4] have been developed and battle tested for processing millions of data records per second in a distributed manner. This step needs to perform several functions.

Data formatting and enrichment - transform messages from the input format into a common format that is accepted by the internal algorithms. Also, additional data fields can be attached, based on background knowledge.

Data aggregation - sometimes we want to measure characteristics of all the data within some time intervals (e.g. average speed in the last 10 minutes).

Data routing - send the transformed and aggregated data to relevant receivers. There may be several listeners for the same type of input data.

3.3.2 Signal detectors

When data is ready for processing, it is routed to signal detectors. They operate on a single data stream, often only on a small partition of it - e.g. single stock, group of related stocks. They handle huge data volumes, so they need to be fast, using very little resources. To achieve great flexibility regarding input data a dynamic allocation of such processors is required. This enables handling of previously unseen data partitions as well as scalability in parallel processing.

These anomalies (signals) have simple models and consequently alert explanations. But they are local in nature - their scope is most often very limited. They also operate on single-data stream, so they don't take into account the anomalies in *"the neighbourhood"*. To overcome this deficiencies, we propose the third step of stream processing, to which signals should be sent.

3.3.3 Incident detectors

This stage of the processing receives signals from different parts of the system, performing scoring of their importance, combining them into *incidents* and thus achieving several advantages.

The scoring algorithm provides option to assign user-guided *subjective importance* to signals - e.g. two statistically equally important anomalies can have completely difference perceived value to the user. This step can also can correlate data across data-streams, a step that is hard to achieve and that proves to be very valuable. Given data from diffrent parts of the system it can create more complex constructs that better evaluate the impact of the current problem on the whole system.

This **level of abstraction** is the main access point for end-users - it more closely follows their way of addressing system malfunctions (e.g "if module A breaks, it will have impact on modules B and C, but module D should remain unaffected").

3.3.4 Background knowledge

To help guide the algorithms during the signal detection we can provide additional background knowledge in different forms, such as metadata, manual thresholds and rules, graphs and other structures. All this data can be used to perform various enhancements of basic algorithms, such as creation of **additional features** in data pre-processing, **up- and down-voting** of results (e.g. estimated impact of detected anomaly), **pruning of search space** in optimization steps, **estimation of affected entities** for given anomaly

or **support for complex simulations** when current performance is measured against historical values. These rules and metadata can be acquired by analyzing historical data as well as collecting knowledge from end-user, e.g. manual feedback/sign-off and active learning.

3.4 Improving actionability

The system modules presented so far are mostly established components that are used also in normal processing steps of modern, cloud-based systems (see [1]). We propose that they should be upgraded with the following functionalities in order to achieve the goal of high-quality actionable alerts, empowering users to manage their complex systems in the most efficient way.

3.4.1 Feedback

Historical incidents are very valuable for learning of informative features that aid detection of anomalies. They are also used for calibrating scoring algorithm that assigns relevance scores to generated signals and incidents. It is common for the organization to require every major detected incident to be manually signed off - a *relevance tag* (e.g. high-relevant, semi-relevant, not-relevant, noise, new-normal) has to be assigned to it by the operators. These tags are used for training of incident-classification algorithms, but we can also construct a more complex setting where a form of backtracking is used to *calibrate* signal detectors.

3.4.2 Active learning

The *active learning* approach [3] can be used to make the manual classification effort more efficient. The system provides untagged examples/incidents where the criteria function returns the value that is the closest to the decision boundary. The user then manually classifies the incident and the classification model is re-trained with this new data. By guiding users in this way the system requires relatively small number of steps to cover the search space and obtain good learning examples.

Our proposed approach incorporates this continuous activity in several areas. GUI module should contain appropriate pages where user can enter his feedback and active-learning input. Storage module contains alerts historical data that can be used for re-training of incident detectors. Storage module also contains old and new incident-detector models that can be picked up automatically by the processing module.

4. VALIDATION AND DISCUSSION

Based on our extensive experience with practical anomaly detection implementation, we identified several new requirements for these systems. The presented approach satisfies them by supporting big-data real-time analytics on one side and actionability via active-learning support on the other.

The system architecture is deployable to cloud environment by design. We also employ modern streaming and storage technologies for transporting and storing of different input data and alerts.

We observed that users appreciate our notion of incidents - a grouping of alerts that occur in certain small time interval. Users feel comfortable with seeing the big picture (an indcident) in then drill down into specific data (individual signal). They reported this feature enables them to cut down time for understanding the problem by an order of magnitude (from hours to minutes).

Active-learning component was well received, as it made manual work more efficient. The users noticed how the algorithm was choosing more and more complex learning examples for manual classification. This helped them feel productive and engadged. They also reported positive impact of active learning on their problem understanding, as they were presented with some potentially problematic situations that went unnoticed in the past.

Based on above observations were conclude that our proposed approach has positively impact on the organization, both for technologies as well as human operators. Additional ideas that were collected from users are listed under future work.

5. CONCLUSIONS AND FUTURE WORK

The focus of our future work is on several advanced scenarios where a lot of added value for users is expected, mixing anomaly detection, optimization and simulation. Main gains are expected to come from feedback collection and active learning. Apart from monitoring IT systems, the target domains are also manufacturing and smart cities. We also collected some features that users commonly inquired about:

- **Root-cause analysis** - when a major incident occurs, many parts of the system get affected. To resolve issues as quickly as possible, the operators should be pointed to the origin of the problem. The anomaly detection system should thus have a capability to point to the first signal with high-impact on the final relevance score.
- **Predictions** - The goal for all monitoring systems is to detect problems as soon as possible. The system must that not only be able to detect signals, but also forecast them, based on past behavior. In order to do that, the algorithms require more metadata and structure to properly model inter-dependencies between signals. Mere observation is much easier than simulation of a complex system with many moving parts. But it is possible and is what users expect from a *modern AI-based system*. Our future reserach will be oriented towards providing and efficiently integrating these functionalities into our anomaly-detection approach.

6. REFERENCES

[1] Anodot anomaly detection system. `http://www.anodot.com`, 2018.

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

[3] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *CoRR*, cs.AI/9603104, 1996.

[4] N. Garg. *Apache Kafka*. Packt Publishing, 2013.

[5] J. Lin. The lambda and the kappa. *IEEE Internet Computing*, 21(5):60–66, 2017.