Scalable Infrastructure and Workflow for Anomaly Detection in an Automotive Industry

1st Anshul Jindal, 2nd Michael Gerndt
Chair of Computer Architecture and Parallel Systems
Technical University of Munich
Garching (near Munich), Germany
anshul.jindal@tum.de, gerndt@in.tum.de

3rd Mario Bauch, 4th Hachim Haddouti
OTD Platform and Services
Munich, Germany

***Abstract*—Anomalies are unexpected instances which significantly deviate from the normal patterns formed by the majority of a dataset. The more an observation deviate from the normal pattern, the more likely it is an anomaly. The continuous increase in the number of car models and configuration possibilities has led to continuous increase in the complexity of logistics supply chain and production. Consequently, it has become difficult to manage the whole IT Landscape, a small anomaly/failure somewhere in the system could lead to a huge loss of money. Therefore, to identify and ultimately resolve quickly a problem in such a system is highly important.**

This paper addresses the *challenge of identifying anomalies in a scalable way*. The new data collected suffers from the problem of lack of labels for training. This challenge is addressed in the developed solution by using multiple unsupervised algorithms and reporting those observation as anomalies which are commonly reported as anomalies by all the algorithms. The developed solution also tackles the problem of data heterogeneity and big size by using Spark underneath for scalable data processing. Scalability test results demonstrate the reduction in training time of 100 transactions by 80% when using 10 cores instead of using 1 core. The results of the study have also pointed out that increasing the number of cores does not necessarily means reduction in the overall execution time, there are other factors like communications between the cores, non-spark related processing tasks, etc which can also influence the execution time.

***Index Terms*—anomaly detection, timeseries, scalable, spark, scalable anomaly detection**

I. Introduction

The automotive industry's car models and derivatives have grown significantly over the years [1]. Due to such an increase, the complexity of the IT Landscape for managing the logistics supply chain and production has grown vastly. The increasing number of interfaces between the systems that run the IT landscape has made the security of the systems a big concern. Therefore, the industry's security team has to frequently patch the systems to avoid any security threat and keep them up to date. However, to keep the overall downtime of the logistics supply chain and production to almost zero, even frequent security patches or updates are not affordable as they will stop the production line and increase downtime. On the other hand, a small problem or a failure somewhere in the IT landscape could disrupt the production line and lead to increased downtime and a huge loss of money. Therefore one needs to detect quickly and solve such failures.

The unexpected instances which significantly deviate from the normal patterns formed by the majority of a dataset are called anomalies. For efficient resolution of an anomaly, an intelligent system is required that can automatically detect anomalies, identify their causes and showcase them in a central user-interface in an intuitive way. This will not only allow the administrators but also the service owners to easily troubleshoot the anomaly, narrow down its root cause and quickly resolve the problem. However, the input data being the time series data imposes a set of challenges for the detection of anomalies due to the lack of defined pattern for the anomaly, noise in the input data and the complexity of detection increases with the length of the time series. Also, for a timestamp, the input data consists of multiple parameter values (called features) which makes the input data multivariate time series data and it is a challenge to identify which parameters are relevant for which anomaly. Also, due to the lack of labels for training, the approach for solving such a problem has to be an unsupervised one. Therefore, our idea to solve this challenge is *to capture normal patterns of the multivariate time series using different unsupervised algorithms and report those observations as anomalies which are commonly reported as anomalies by all the algorithms*. It is very highly likely that an observation reported as an anomaly by all the algorithms is also actually an anomaly.

This work is partnered with an automotive industry, where the environment contains data from different sources such as supply chain, the production pipeline as well as from the IT systems. Out of these, IT systems of the production environment data is used in this work. There are over 100 thousand requests to these systems at any instance of time and these requests are called transactions. Each transaction has over 50 features or parameters such as the total response time, CPU utilization and database query time and these parameters values are collected every five minutes. It is time inefficient to find the anomalies without using a distributed data processing framework because of the size of the data, therefore the challenge here is to build a scalable infrastructure and framework for the anomaly detection with all the performance requirements, and costs taken into consideration. Our approach to solving this problem is *to build the solution using the Spark framework underneath for distributed data processing and scalability*. The third challenge as part of this work is to provide an intuitive and easy way to view the anomalies and provide a way for the system administrator or the service owner to answer the question why an observation is detected

as an anomaly. Anomaly interpretation can help in efficient troubleshooting, and thus is often required in practice [2]. Our solution to this problem is *to provide the user with the overall summary heat map for viewing the anomalies and also provide a list of relevant feature names from all the features which have the highest correlation with an anomaly observation.*

The contributions of this paper are summarized as follows:

- We propose an approach using multiple unsupervised algorithms along with a Spark-based framework for building a scalable infrastructure and framework for anomaly detection in the automotive industry.
- We also present an intuitive and easy way for interpreting the anomalies by providing the heat map of the anomalies along with the list of relevant feature names having the highest correlation with the anomaly.

The rest of this paper is composed as follows. Section 2 discusses background knowledge. Section 3 studies the related works. Section 4 describes the overall problem statement and the followed approach to solve it. Section 5 presents the implementation details of the developed system. Section 6 provides experimental configuration details along with results of the conducted analysis. Section 7 summarizes the results and lastly, Section 8 concludes the paper.

II. Background

In this section, we present the basic background knowledge used in this paper.

A. Storage Frameworks

1) TimescaleDB: TimescaleDB is an open-source time-series database optimized for fast ingest and complex queries [3]. TimescaleDB is implemented as an extension on PostgreSQL and runs within the PostgreSQL instance. This extension model of TimescaleDB over PostgreSQL allows the database to take advantage of many of the attributes of PostgreSQL and also at the same time, it leverages the high degree of customization available to the extensions by adding hooks deep into PostgreSQL's query planner, data model, and execution engine. It exposes singular tables called as hypertables, which are a virtual view of many individual tables holding the data, called chunks. All the hypertables are partitioned by time interval, and can further be partitioned by a key such as transaction ID, system id, etc. These partitions are disjoint, which helps to minimize the set of chunks to read for resolving a query. TimescaleDB is used as the main database in the developed system for storing all the data.

2) Apache Parquet: It is an open-source column-oriented data storage format of the Apache Hadoop ecosystem [4]. It provides an efficient way for data compression and encoding schemes with the enhanced performance for handling data in bulk. Parquet stores binary data in a column-oriented way, where the values of each column are adjacent, enabling better compression. It is good for queries which require to read some columns from a wide number of columns in a table as only required columns are read and the number of I/Os are minimized [5]. Also, aggregation queries are less time consuming in Parquet as compared to row-oriented databases (csv files). In our research, all the input data resides in different csv files, thus, before any data analysis the csv files are read and stored in a single Parquet file for efficient processing.

B. Unsupervised Anomaly Detection Algorithms

In this subsection, we briefly describe about the different unsupervised anomaly detection algorithms used in this work.

1) Local Outlier Factor (LOF): The Local Outlier Factor algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers outliers as the samples that have a substantially lower density than their neighbors [6], [7]. The local outlier factor is based on a concept of a local density, where locality is given by "k" nearest neighbors, whose distance is used to estimate the density. The LOF is the average ratio of the Local Reachability Density (It tells how far one has to travel from the point to reach the next point or cluster of points) of the neighbors of "x" to the LRD of "x". If the ratio is greater than 1, the density of point "x" is smaller than the density of its neighbors and, thus, from point "x", we have to travel longer distance to get to the next point or cluster of points, therefore "x" is an outlier. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be outliers.

2) Isolation Forest (IF): It is also an unsupervised anomaly detection algorithm, which is based on the basis of decision trees. It is based on the principle that, outliers are less frequent than the regular observations and are different from them in terms of values (means they lie further away from the normal observations in the feature space) [8]. It create partitions by first randomly selecting a "feature" and then randomly selecting a split value between the minimum and the maximum of the selected feature. It can be represented by a tree structure. By doing such a partitioning, anomalies/outliers should be located closer to the root of the tree with fewer splits necessary. As part of this algorithm, each observation is given a score between 0 and 1 and if the score lies close to 1 then the observation is an anomaly.

3) One-Class SVM (OCSVM): Support vector machines (SVMs) are supervised learning models that analyze data and recognize patterns, and can be used for both classification and regression tasks. Typically, the SVM algorithm is given a set of training examples labeled as belonging to one of the two classes. An SVM model is based on dividing the training sample points into separate categories by as wide a gap as possible, while penalizing training samples that fall on the wrong side of the gap. The SVM model then makes predictions by assigning points to one side of the gap or the other [9]. In one-class SVM, the support vector model is trained on the data that has only one class, which is the "normal" class. It infers the properties of normal cases and from these properties the model can predict which observations are unlike the normal observations [10].

4) Z-score: The Z-score, or standard score, is a way of describing a data point in terms of its relationship to the mean and standard deviation of a group of points. Taking a Z-score is simply mapping the data onto a distribution whose mean (μ) is defined as 0 and whose standard deviation (σ) is defined as 1. The intuition behind the Z-score method is that, anything that is too far from zero (for example 3σ or -3σ) is considered as an outlier. Z-score being an univariate method makes it difficult to use on the input multivariate data. Therefore in this work, it is used on univariate scores data which are given as output from the above-discussed algorithms for anomaly detection.

III. Related work

The outlined challenge of anomaly detection was considered by the researchers from different perspectives. Several regression-based techniques for time-series modeling such as robust regression [11], auto-regressive models [12], auto-regressive moving average (ARMA) models [13], [14], and auto-regressive integrated moving average (ARIMA) models [15], have been developed for anomaly detection. Many deep learning models have also been proposed for detecting anomalies within univariate and multivariate time series data which seems to be quite effective in detection [16]–[18]. However, choosing the right method and model varies from time series to time series due to dissimilarities between them.

The models used for learning the normalized behavior of the data take a long time to train thus a scalable way is required. There already exists some research in this area. For example Lee et. al has proposed a scalable framework for anomaly detection in software-defined networks [19] and Smith et. al has developed a framework for detecting anomalies in the large Cloud systems [20]. However, these systems differ from the automotive industry's IT Landscape data and thus cannot be directly used. Therefore, this work focuses on developing a framework for anomaly detection in a scalable way using the automotive industry's IT Landscape data.

IV. Methodology

In this section, we present the problem statement of anomaly detection in detail and introduce the overall followed approach.

A. Problem Statement

A time series data contains successive observations which are collected at fixed timestamps. In our study, we focus on multiple transactions data, defined as $T = \{T_1, T_2, ..., T_K\}$, where K is the number of transactions, and a transaction T_i ($i \leq K$) represents a multi-variate time series data defined as $x = \{x_1, x_2, ..., x_N\}$, where N is the length of x and an observation $x_t \in R^M$ is an M-dimensional vector at time t ($t \leq N$): $x_t = \{x_t^1, x_t^2, ..., x_t^M\}$, and $x \in R^{M \times N}$, where M is the number of features. A time series window of length L referred by $x_{t-L:t} \in R^{M \times (L+1}$ is used to denote a sequence of observations $\{x_{t-L}, x_{t-L+1}, ..., x_t\}$ from time $t - L$ to t.

The objective of this work is to determine whether an observation x_t is an anomalous or not along with the list of most relevant feature names F ($F \leq M$) for the anomaly interpretation. Also, the system to do so should be scalable with the number of transactions.

B. Approach

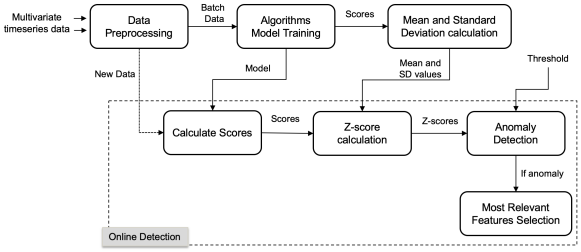

Fig. 1: Overall followed approach workflow

Fig. 1 shows the overall workflow of the followed approach. The whole system consists of two sub-phases: offline training and online detection. Data preprocessing is a module shared by both the phases. During data preprocessing, the dataset is transformed by resampling the number of observations to one every defined resampling time resolution. For time series modeling, historical values are beneficial for understanding current data. Therefore, after preprocessing, a sequence of observations $x_{t-L:t}$ instead of just x_t belonging to a transaction T_i is used as the training data and is sent for different algorithms "a" belonging to **LOF, IF and OCSVM**, for model training. Each algorithm gives scores as output for each of the training observation in a transaction T_i represented by $s^a = \{s^a_{t-L}, s^a_{t-L+1}, ..., s^a_t\}$. Further, mean and standard deviation values represented by μ^a_i and σ^a_i, are calculated per algorithm a and per transaction T_i on these output scores and saved to the disk. This offline training procedure can be conducted routinely, e.g., once per day or week.

In the Online Detection phase, for a new observation x_t belonging to a transaction T_i, the Calculate Scores module uses the stored trained model per algorithm "a" and determines the score for it s^a_i. Afterwards, Z-score Z^a_i per algorithm is calculated for x_t using the stored mean μ^a_i and standard deviation σ^a_i. The calculated Z-scores for x_t for all the three algorithms, are used to determine whether it is an anomaly or not. If all the Z-scores of x_t are above a threshold (usually 3σ), x_t will be regarded as anomalous, otherwise, it is normal. If x_t is detected as an anomaly, then the most relevant features F ($F \leq M$) for interpreting the anomaly are determined.

To encounter the problem of big data, the whole system is built upon the Spark infrastructure for the distributed data processing and scalability of the system on demand.

V. Implementation

In this section, we first present the overall architecture of the developed system, followed by description of its components.

A. Overall Architecture

The developed tool is mainly written in Python and PySpark along with HTML, CSS, and JavaScript for the web-interface. The developed system automates the setup of a Spark cluster and deploys the other components in that cluster. Being the Spark-based development, the data processing and analysis can

be scaled easily depending upon the requirement and availability of resources. The overall execution workflow between its components in a typical use-case is shown in Fig. 2.

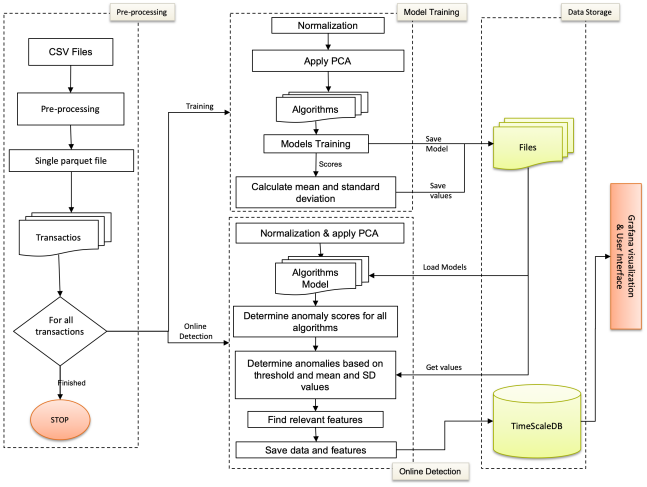

Fig. 2: Overall execution pipeline of anomaly detection

The procedure starts with the user providing the path where all the CSV files belonging to the transactions are situated through the user-interface and then, these CSV files are combined and stored into a single parquet file. This allows fast access of the data for further analysis due to the column-major storage of the parquet file format. Afterward, some preprocessing steps are conducted. These steps are conducted both for the training as well as the new data. Subsequently, models are trained separately for each transaction's training data and saved as files to the disk. Each algorithm gives a score as an output for each training observation. Separate per algorithm and per transaction, mean and standard deviation values are also computed on all the training observation's scores and saved to the disk.

After training, each algorithm's saved model for a transaction T_i is used to determine the score on a new observation x_t. Afterward, each model's score on this new observation along with the model's stored mean μ_i^a and standard deviation σ_i^a is used to calculate the Z-score Z_i^a per algorithm "a" for x_t. If all the calculated Z-scores are above a threshold, x_t will be regarded as anomalous, otherwise, it is normal. If x_t is detected as an anomaly, then the 10 most relevant features F ($F \leq M$) are determined and stored together with the anomaly labeled data into the database. Finally, the stored data is queried and visualized in Grafana.

B. Preprocessing

The first step towards the detection of anomalies is to find the right data to analyze. All the data belonging to different transactions are in the CSV files and the initial challenge is to efficiently group the records based on the transactions to which they belong so that a model per transaction can be trained. Therefore, firstly some preprocessing steps like conversion of string timestamp to Python datetime format and conversion of string numbers to float are done and then all CSV files are combined and stored into a single parquet file for column-major storage. We have found that directly processing CSV files is much slower in comparison to processing a single parquet file. This can be due to the column-major storage of the files in the parquet format, allowing faster aggregation queries and efficiently processing. Afterward, the records belonging to a transaction are grouped efficiently using the PySpark groupBy method. Each processed transaction data is then further passed to the next component.

C. Models Training

Each pre-processed transaction data is read as Python pandas dataframe. The length of each input sequence training dataframe (e.g., $x_{t-L:t}$) is L +1 and the decision to choose it is left to the user. First, the data is normalized using the python sklearn's standard scalar which removes the mean from each observation and scales it to unit variance. Afterward, Principle Component Analysis (PCA) is applied to this normalized data to reduce the number of dimensions. The models of all three algorithms are trained on the reduced data. The default parameters used for training are mentioned in the Experimental Configuration subsection. The trained models are saved to the disk. These models output a score for each observation and this score for all the training observations is used to calculate mean score μ_i^a and standard deviation σ_i^a per algorithm model. These values are used to decide whether a new observation is an anomaly or not in the online anomaly detection phase.

D. Online Anomaly Detection

Now we can determine whether an observation at a time step (t, denoted as x_t) belonging to a transaction T_i is an anomalous or not using the trained models. Each of the three algorithm's model is loaded and its predict function is called on x_t to get a score for it. Afterward, each algorithm model's score on this new observation along with the its stored mean μ_i^a and standard deviation σ_i^a is used to calculate the Z-score Z_i^a per algorithm "a" model for x_t. The calculated Z-scores for x_t are used to determine whether it is an anomaly or not. If the Z-scores of all the models are above the threshold then x_t is marked as an anomaly otherwise, not. This threshold is given a default value of 4σ but it can be changed from the user interface. If the observation is marked as an anomaly then its correlation is calculated with all the features. The top 10 highly correlated features marked as the most relevant features along with anomaly labeled data is saved in the database.

E. Visualization

For visualizing the data, different SQL queries based panels are integrated into the Grafana dashboard :

1) Overall anomalies heat map: This panel is responsible for showcasing a heat map of the overall anomalies present in the top 10 transactions. These top 10 transactions are the transactions with the most number of anomalies in the configured time period. Selecting the number of transactions (different from 10) can also be configured. This panel displays a heat map with x-axis representing the time, y-axis representing

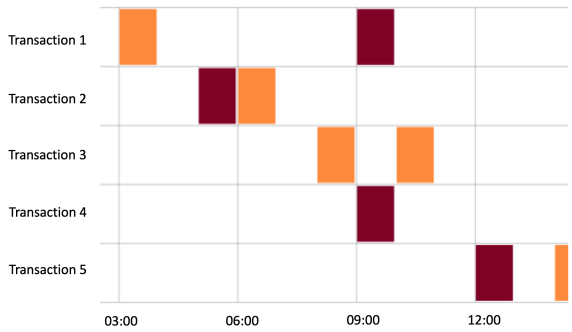

Fig. 3: Anomalies heat map showcased in 5 transactions

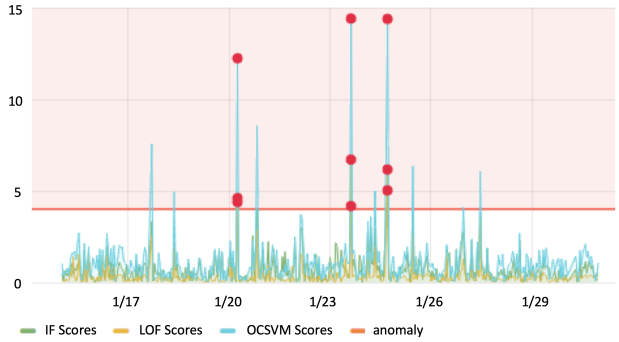

Fig. 4: Algorithm's scores with highlighted ones above 4σ

different transactions and count of anomalies in a transaction is represented using different colours.

2) A transaction's anomalies heat map: This panel in the Grafana dashboard displays a heat map of the anomalies present in a single transaction. It is similar to the previous one but instead of showcasing 10 transaction, the idea here is to show only one with a detail view.

3) Anomalies view in different features: Interpreting the anomalies or finding out the root cause of the anomalies in a transaction is very important, as it will allow the system administrator to troubleshoot the problem. Therefore, knowing the feature set which has the highest correlation with the anomalies will help administrator narrow down the root cause. This panel showcase the anomalies in the most relevant features set along with their values.

VI. Results and Analysis

A. Experimental Configuration

All the analyses and tests are conducted using the dataset provided by our partnered industry and is not disclosed due to the security concerns. Different hyper-parameters set in our experiments are as follows. The length of the input data sequence used for training is set to first 15 days and the testing set size is last 15 days of the January month. For all the algorithms, the default outliers fraction is set to 0.001. In the case of the Isolation forest, the number of estimators is set to 100 while for other parameters default values are used. For LOF, all the parameter's default value is used and for OCSVM, the kernel used is RBF with gamma value set to 0.01. The default threshold is set to 4σ. The Spark driver's and executor's memory is set to 10GB and the number of cores in the Spark cluster is set to 10 from the available 12 cores.

B. Results

Fig. 3 shows a graphic of the heat map showcasing the anomalies present in the 5 transactions on a day in the January month. These 5 transactions (names hashed for data security) are the transactions with the most number of anomalies in the configured time period. Time is represented on the x-axis, the y-axis shows different transactions and count of anomalies in a transaction is represented using different colors. The dark red color corresponds to the transaction and time when there are more anomalies, therefore one can click on it and see a detailed view of it for further troubleshooting.

Fig. 4 shows a different algorithm's output scores on one of the transactions. One can see that all three algorithms result in high scores (above 4σ) for some of the observations which are highlighted as anomalies. Also, LOF and IF have relatively smaller scores for the anomaly points as compared to the OCSVM. In general, OCSVM results in higher scores for other observations as well but are not highlighted because only those are selected where all the algorithm's scores are above 4σ.

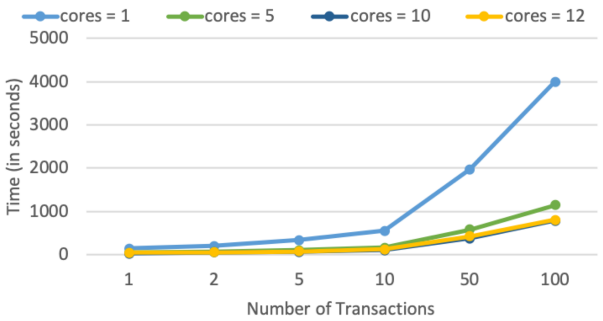

Fig. 5: Training time for different number of transactions and cores.

We further have conducted the developed system's scalability test by recording the time required to train different number of transactions with different number of cores and results are shown in the Table I and Fig. 5. One can see that the time required to train different number of transactions reduced significantly when using 10 cores. For training 100 transactions the time taken with 10 cores was reduced by 80 percent as compared to when using 1 core. However, with 12 cores (all the available cores) the training time in all the tests increased a little bit as compared to when using 10 cores. This could be either due to the increase in communication between the cores or because there are no remaining cores for handling other non-spark related processing as a result the system waits for the cores to get free and thus increases overall processing time. Thus, in the current environment, the optimal number of cores is 10 out of the available 12 cores.

VII. Discussion

Usage of Parquet and TimescaleDB as the underneath storage framework offers a promising technique for the storage and efficient data queries. Processing big data require high cost

TABLE I: Training time required in seconds for different number of transactions with different number of cores

No. of Transactions	Cores=1	Cores=5	Cores=10	Cores=12
1	142	47	29.4	42.1
2	200	62	51	53.7
5	338	107	69	74
10	551	162	106	123
50	1977	579	373	424
100	4004	1142	787	804

for storage and computation therefore, storing the files in this fashion serves efficiency and performance in both storage and processing. Scalability test results demonstrate that increasing the number of cores for data processing can decrease the overall processing time as in our tests reducing the training time of 100 transactions by 80% when using 10 cores instead of using 1 core. However, simply increasing the number of cores continuously does not help in the reduction of overall processing time. There are other factors like communications between the cores, non-spark related processing tasks, etc which can influence the processing time. Thus, one needs to find the optimal number of cores for their environment.

The combination of three algorithms result is used for pointing out whether an observation is an anomaly or not. The intuition behind this approach is that it is highly likely that an observation reported as an anomaly by all the algorithms is also actually an anomaly. The training performance does suffer a little from training three models however, the detection part which is mainly considered as the performance metric is quite fast. The observations detected as anomalies are the ones having Z-scores in all the algorithms above a given threshold. This allows to detect the anomalies with the scores deviating highly from the normal score patterns in all algorithms and offers a novel technique for anomaly detection. Also, using the correlation between the anomalies detected and the features gives an intuitive way for interpreting and troubleshooting the anomalies by providing the most relevant feature set.

VIII. Conclusion

Anomaly detection in the automotive industry can greatly help the system administrators and the service owners to discover and troubleshoot abnormal behaviors and prevent the huge loss of money. In this paper, we present the implementation details of the tool that was used in our partnered industry environment for building a scalable infrastructure and framework for anomaly detection. We believe that the method adopted using Spark underneath allows scalable and efficient processing. Furthermore, using the PostgreSQL with TimescaleDB for storing the data makes sure that queries for visualization are fast. Also, the presented novel approach of using together multiple algorithms for anomaly detection gives high confidence in the resulting anomalies.

Prospective directions of future work include analysis of the accuracy of the results along with the addition of deep learning and online learning-based approaches for anomaly detection.

References

[1] G. Trade and Invest, "The automotive industry in german¥," [Online; Accessed: 17-November-2019]. [Online]. Available: https://www.gtai.de/GTAI/Content/EN/Invest/_SharedDocs/Downloads/GTAI/Industry-overviews/industry-overview-automotive-industry-en.pdf

[2] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: ACM, 2018, pp. 387–395. [Online]. Available: http://doi.acm.org/10.1145/3219819.3219845

[3] "Timescaledb overview," [Online; Accessed: 1-October-2019]. [Online]. Available: https://docs.timescale.com/latest/introduction

[4] "Apache parquet," [Online; Accessed: 10-October-2019]. [Online]. Available: https://parquet.apache.org/documentation/latest/

[5] R. Brundesh, "Parquet file format hadoop," June 2016, [Online; Accessed: 10-October-2019]. [Online]. Available: https://acadgild.com/blog/parquet-file-format-hadoop

[6] "Outlier detection with local outlier factor (lof)," [Online; Accessed: 1-October-2019]. [Online]. Available: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 93–104. [Online]. Available: http://doi.acm.org/10.1145/342009.335388

[8] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, Dec 2008, pp. 413–422.

[9] "One-class support vector machine," [Online; Accessed: 1-October-2019]. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/one-class-support-vector-machine

[10] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 582–588. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009740

[11] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. New York, NY, USA: John Wiley & Sons, Inc., 1987.

[12] A. J. Fox, "Outliers in time series," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 34, no. 3, pp. 350–363, 1972. [Online]. Available: http://www.jstor.org/stable/2985071

[13] B. Abraham and G. E. P. Box, "Bayesian analysis of some outlier problems in time series," *Biometrika*, vol. 66, no. 2, pp. 229–236, 1979. [Online]. Available: http://www.jstor.org/stable/2335653

[14] B. Abraham and A. Chuang, "Outlier detection and time series modeling," *Technometrics*, vol. 31, no. 2, pp. 241–248, May 1989. [Online]. Available: http://dx.doi.org/10.2307/1268821

[15] A. M. Bianco, M. García Ben, E. J. Martínez, and V. J. Yohai, "Outlier detection in regression models with arima errors using robust estimates," *Journal of Forecasting*, vol. 20, no. 8, pp. 565–579, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.768

[16] D. Li, D. Chen, L. Shi, B. Jin, J. Goh, and S.-K. Ng, "Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks," in *ICANN*, 2019.

[17] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: ACM, 2019, pp. 2828–2837. [Online]. Available: http://doi.acm.org/10.1145/3292500.3330672

[18] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber," *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2017. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2017.19

[19] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2017, pp. 249–260.

[20] D. Smith, Q. Guan, and S. Fu, "An anomaly detection framework for autonomic management of compute cloud systems," in *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, July 2010, pp. 376–381.