

ANOMALY DETECTION IN STREAM DATA PROCESSING IN REAL TIME

¹MAXIM DUNAEV, ²KONSTANTIN ZAYTSEV, ³ROMAN ELCHENKOV, ⁴DENIS SAVITSKY
^{1,2,3,4}National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), 31 Kashirskoe
Avenue, Moscow, 115409, Russia
E-mail: kszajtsev@gmail.com

ABSTRACT

The purpose of the present work is to study methods for solving problems of anomaly detection and prediction of time series values when processing streaming data in real-time in a network environment and their improvement. To solve this problem the authors propose a Real-Time K-Means modification with preliminary markup. The effectiveness of the modification is confirmed by comparing it with the frequently used Streaming K-Means from the Apache Spark Mllib library. To solve the problem of predicting time series when processing streaming data in real-time, the authors propose a modification of the autoregression model with a given AR order by adding the inheritance function of the previous values of the time series to it. The results of comparative experiments of the proposed Real-Time AR modification with classical AR confirmed the effectiveness of the modification, which is especially evident in the presence of anomalies in the behavior of the time series. The proposed algorithm modifications allow not only parallelizing calculations using the deferred computing paradigm but also configuring the model fleetingly in the Apache Spark ecosystem. To conduct experiments with algorithms, a dataset was built – a data slice from 1,000 measurements of the Apache Kafka server metrics log with one topic, two producers, and one consumer. Anomalous fragments were artificially added to the dataset, differing by a large number of messages per second and/or message size. The values of the original dataset were normalized and shifted to the average value of the training fetch. Moreover, static and highly correlated metrics were eliminated. The results of the application of the developed algorithms in solving the problems of detecting and predicting the values of time series have shown that even the presence of behavior anomalies does not distort predictions significantly.

Keywords: *Logs, Technological Platform, Machine Learning, Deep Learning, Apache Software Foundation.*

1. INTRODUCTION

Today, every major IT company works with dozens or even hundreds of services deployed simultaneously. The only reliable source of information, characterizing the state of applications is service logs – a set of metrics, represented by time series of values. Using machine learning methods and the values of log metrics, one can track various service failures, attempts to interfere with the operation of the application, acts of fraud, as well as classify anomalies in the behavior of services, and predict the values of log metrics for the nearest time period. This knowledge helps in risk management and decision-making.

To solve the above problems, several successful approaches are available which are considered to be classic. Thus, for analyzing time series, one can distinguish algorithms such as statistical ones (ARMA, ARIMA, GARCH, Holt-Winters model, etc.), linear regression algorithms

[1], as well as neural networks with different architectures (RNN, LSTM, etc.) [2, 3]. To detect anomalies, algorithms, such as K-Means or KNN are usually used [4-6].

These machine learning methods, undoubtedly, can solve the above-mentioned problems of managing complex objects, and are actively used for various activities. However, their straight application is not suitable for working with continuous streams of high-intensity big data in real-time. The standard approach for any supervised learning task requires training the model based on a pre-prepared training fetch, testing the correctness of predictions on a test fetch, and then interpolating the results to a wider fetch of real data. When processing data, represented by time series, this approach can be applied with serious limitations, since, firstly, time-series data are rigidly ordered, and secondly, these data are continuously changing over time. Therefore, it is not reasonable to use models previously trained on outdated data. An example is the inability to use

a model based on a fetch of a load of an online store focused on traffic of 2,000 users per day, after a short-term increase in traffic due to an advertising campaign to 3,000 customers per day. Traffic is changing and, possibly, the social status of buyers with other needs is changing as well, and therefore using a previously built model will give inaccurate results at best, and completely incorrect outcomes at worst.

One can try to retrain the model at certain time intervals. However, such a solution is ineffective because there is usually a lot of processed data, and several models can be deployed simultaneously to work with different time series. Therefore, the process of frequent model changes will be associated with fairly high costs. But if it is important for us to constantly retrain the data analysis model during operation, then we can only use algorithms that retrain faster than the next batch of training fetch data arrives from the stream. It is customary to say about such algorithms that they are trained on the fly. A class of such models is discussed in publications as part of online learning. For high-speed, high-intensity work, it is required to use frameworks that allow distributed computing over large amounts of data. One of the most popular frameworks is Apache Spark, designed for distributed processing of structured and unstructured data, and being a part of the Hadoop ecosystem [7, 8].

Besides classical approaches, today scholars are increasingly looking for real-time models and algorithms for solving problems of detecting anomalies and predicting time series values.

Thus, in the article [9], a real-time implementation of ARIMA and even an open-source library for anomaly detection and prediction using real-time ARIMA models are proposed. However, the proposed framework cannot operate in a distributed environment.

To search for anomalies, the authors [10] suggest using OPTICS, but for datasets with significant value dynamics, an attempt to use even improved DBScan OPTICS does not allow detecting anomalies. Researchers [11] note the complexity of simulating prediction processes with high accuracy and sensitivity of anomalous states in real-time. To solve this problem, CF-LSTM is proposed, representing an improved multidimensional entropy transfer method for estimating physical causality between multidimensional time series in combination with the LSTM model. For thousands of time series, some of which are independent, it is

hardly possible to assess physical causality, not to mention the accuracy and sensitivity of forecasting.

The article [12] proposes a comprehensive method for detecting anomalies in telecommunications traffic, which includes components based on entropy analysis, signature analysis, and machine learning using fractal and recurrent analysis. The article assumes that combining different approaches will give an improved result. This is true only for static areas. If it is necessary to frequently change the set of models used, this approach will lead to an information collapse.

The authors [13] describe a new application of robust estimation for detecting volumetric anomalies in the traffic of a computer network in real-time. The proposed tests are based on the location and variance of the sample and are derived from relatively unknown zero-order statistics. This approach is only effective for detecting one type of failure with heavy tails outside of network traffic and cannot be used for other anomalies.

The purpose of the present work is to study existing methods for solving problems of detecting anomalies by the log-based metrics of technological platforms and predicting the time series values when processing real-time streaming data in a network environment and the possibilities for their improvement.

To verify the solutions proposed by the authors, it is necessary to conduct comparative experiments with the most common algorithms used to solve the set problems on prepared datasets containing anomalies. Experiments will allow determining in practice the effectiveness of the selected methods for solving the set problems.

2. MATERIALS AND METHODS

2.1 Problem Statement

Abnormal data on the operation of technological platforms usually include outliers of individual log metrics values that do not correspond to the accepted logic of the processes. Often abnormal data characterizes some kind of problem, for example, a structural defect, attempted fraud, etc., although sometimes it is impossible to determine the cause of the outlier of the characteristic. Two problems arise in connection with anomalies:

- real-time anomaly detection,
- prediction of possible anomalies.

2.2 Overview of Problem-Solving Algorithms

2.2.1 Algorithms used for anomaly detection

Let us consider two algorithms – K-means and Streaming K-means.

a) K-means

The problem of detecting anomalies can be considered as a clustering problem. Let there be n clusters of normal behavior and one cluster of abnormal behavior. The method of its determination will be given later in this article. At the first stage, we build a model based on historical behavioral data, which we will evaluate based on the assumptions of experts. At the second stage, we will measure distances online within clusters for new data appearing in the log, determining their anomaly, while periodically updating the model.

K-means is one of the machine learning algorithms that solve the clustering problem. This algorithm is non-hierarchical and iterative. It has gained great popularity due to its simplicity, clarity

of implementation and fairly high quality of the results obtained.

The main idea of the algorithm is, as known, that at the regular i -th step of the iterative procedure, the center of mass of each cluster obtained at the previous $(i-1)$ -th step is calculated anew, then the vectors are divided into clusters again according to which of the new centers turned out to be closer to them by the selected metrics.

The algorithm terminates when there is no change in the intracluster distance at the next iteration. The number of such iterations is finite, since the number of possible partitions of a finite set is finite, and at each step, the total square deviation (V) decreases. Therefore, there will be no looping.

For certainty, we give the pseudocode of this algorithm:

1. Setting the initial position of cluster centers $\mu_y, y \in Y$.
2. Assigning each x_i to the nearest center:

$$y_i := \arg \min p(x_i, \mu_y), i = 1, \dots, k, \tag{1}$$

where x_i and y_i are the coordinate and the cluster to which point i should be assigned.

3. Calculating the new positions of the centers as the average of the coordinates of the points assigned to cluster i :

$$\mu_{y_j} := \frac{\sum_{i=1}^l [y_i = y] f_j(x_i)}{\sum_{i=1}^l [y_i = y]}, y \in Y, j = 1, \dots, n \tag{2}$$

Repeating steps 2 and 3 until the composition of y_i clusters stops changing [14].

b) Streaming K-means

The Streaming K-means algorithm interacts with the coming streaming data, updating meta-information about clusters dynamically as new data is received with the ability to control the forgetting of old data (parameter α in formula 3) [15]. This algorithm uses generalized rules for updating K-Means clusters by small data packets. For each data packet, the nearest cluster is assigned to each point, and then new cluster centers are calculated:

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t} \tag{3}$$

$$n_{t+1} = n_t + m_t, \tag{4}$$

where c_t is the previous cluster center, n_t is the number of points in cluster t at the current moment, x_t is the new cluster center made up of points received in the package and assigned to cluster t , m_t is the number of points added to the cluster.

The attenuation criterion α , specified within the limits $[0, 1]$, is used to regulate the degree of "forgetting" of past values. Thus, at $\alpha=1$, all data from the beginning of the observation will be used, at $\alpha=0$, all data except the last ones will be discarded. This parameter is similar to an exponentially weighted moving average.

The attenuation can also be set by the halfLife parameter (half-life), which determines the correct attenuation coefficient α so that for data received at time moment t , their contribution by time point $t+\text{halfLife}$ drops to 0.5.

2.2.2 Algorithms for predicting time series values

To predict the time series values, we also consider two algorithms.

a) Autoregressive model

This is a time series model in which the time series values at the current time point depend linearly on the previous values [16]. This model can be represented by an equation of the form:

$$X_t = c + \sum_{i=1}^p a_i X_{t-i} + \varepsilon_t \tag{5}$$

where a_i are the autoregression coefficients, c is a constant coefficient, ε_t is stationary noise, i.e. a sequence of random variables distributed, as a rule, according to a normal law with an average equal to zero, p is the order of the model.

The optimization problem is to determine both the parameter p , i.e. the number of previous values, used to predict the current value of the time series and the autoregression coefficients.

b) ARMA

The moving average autoregression model is a mathematical model that is used to predict the values of a stationary time series variable.

The ARMA (p, q) model, where p and q are integers, specifying the order of the model, is the process of generating a time series of the form [17]:

$$X_t = c + \varepsilon_t + \sum_{i=1}^p a_i X_{t-i} + \sum_{i=1}^q \beta_i \varepsilon_{t-i} \tag{6}$$

where c is a constant, ε_t is stationary noise, a_i and β_i are autoregressive coefficients and moving average coefficients.

The optimization problem is to determine the order of the model (the p and q numbers) and the a_i and β_i coefficients.

2.3 Preparing Data for Experiments

To test and compare the results of the constructed models, a system with the architecture, shown in Figure 1, was deployed locally.

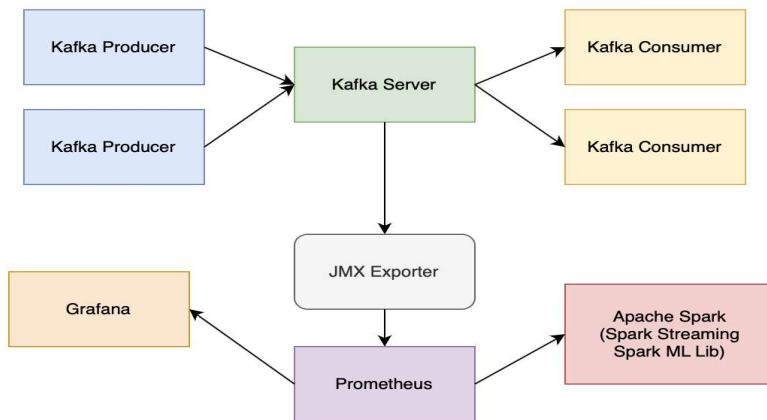


Figure 1: System Architecture

To prepare experimental data, the metrics of the logs of the Apache Kafka Open Source product are used. Kafka producers were programs, written in the high-level Python programming language that send messages with a certain frequency to predefined Kafka topics. The same

programs generated anomalies, represented by a stream of messages with the number of messages per second exceeding the average value by 2,500 times compared to a similar metric for any other time period.

Two producers were used (Table 1).

Table 1: Producers 1 and 2 message generation plans

| Producer 1 | Producer 2 |
|--|---|
| generates messages with a length of 70 bytes and sends them according to plan: <ul style="list-style-type: none"> • one message every two seconds, • two messages every ten seconds, • five messages every fiftieth second, • one hundred messages every five minutes and twenty seconds | generates messages of variable length, sends them, and additionally generates two abnormal messages according to the plan: <ul style="list-style-type: none"> • one message of 70 bytes in length every two seconds, • two abnormal messages with a length of 960 kilobytes |

The resulting dataset is a set of more than two thousand time-series – metrics, collected by the JMX-exporter in the amount of 1,000 rows. Some metrics, uncorrelated or poorly correlated with the number of messages per second, such as `jvm_threads_state`, `jmx_scrape_duration_seconds`, etc. were excluded from the dataset. The data were normalized and shifted relative to the average of the training fetch.

Note that the study does not address the issues of emerging new metrics (when a new topic is introduced in Kafka) and the irrelevance of old metrics (when removing topics or individual Kafka consumers/producers). They were not considered because these processes are purely technical.

In real conditions, data will come from the real-time application of monitoring services, however, for research purposes it is sufficient to use a pre-prepared dataset coming in streaming mode, for example, using Apache Streaming.

2.4 Anomaly Detection

A new dataset that underwent additional preprocessing was prepared from the original one to detect anomalies. The data in the new dataset were normalized, as well as static metrics and correlating metrics were removed.

The Spark MLlib library includes a Streaming K-Means model that can be used for streaming data clustering. However, this model has a disadvantage: it is impossible to load a prepared model into it, and therefore it always comes in action at a cold start. This inconvenience of the algorithm was eliminated by adding an adapter (Figure 2) between the usual K-Means model, which is used to pre-train the model on a certain amount of historical data, and the self-written K-Means Streaming model, which works according to formulas (3) and (4).

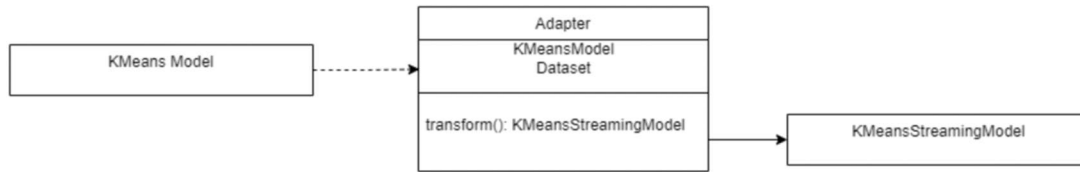


Figure 2: Class Diagram of the Implemented Adapter Between KMeans and Streaming KMeans

This is implemented as follows: after pre-training using K-Means, K-Means Streaming loads the model metadata, such as the number of clusters and their centers. Then, using the dataset on which the model was trained, it determines the number of points in each cluster and saves them.

2.5 Prediction of Time Series Values

To solve the problem of predicting the values of time series, several of the most informative metrics of the Kafka product were selected (Table 2).

Table 2: Kafka Metrics Used

| Name | Meaning |
|---|--|
| <code>kafka_consumer_records_consumed_rate</code> | The average number of records consumed per second |
| <code>kafka_consumer_bytes_consumed_rate</code> | The average number of bytes consumed by consumers per second |
| <code>kafka_consumer_fetch_rate</code> | The number of fetch requests per second |
| <code>kafka_consumer_fetch_latency_max</code> | The max time, taken for a fetch request |

The solution to this problem is illustrated below using the example of the `kafka_consumer_bytes_consumed_rate` metric, which shows the average number of bytes consumed per second. This metric strongly correlates with the

number of messages per second, which, with an increase in the volume of incoming traffic, allows it to be used in the current task. The initial and normalized views of this metric are shown in Figures 3 and 4.

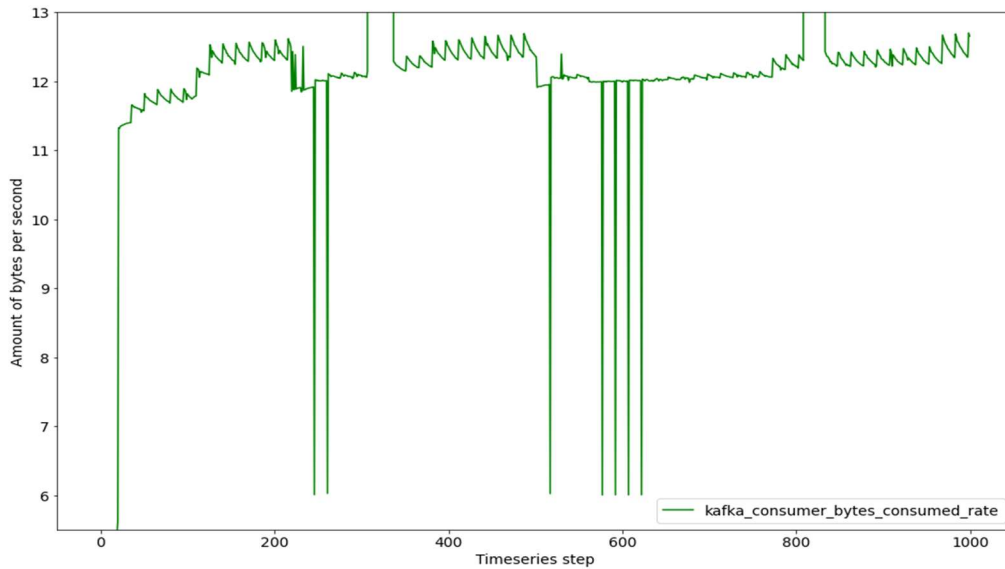


Figure 3: The Data of the Kafka_Consumer_Bytes_Consumed_Rate metric

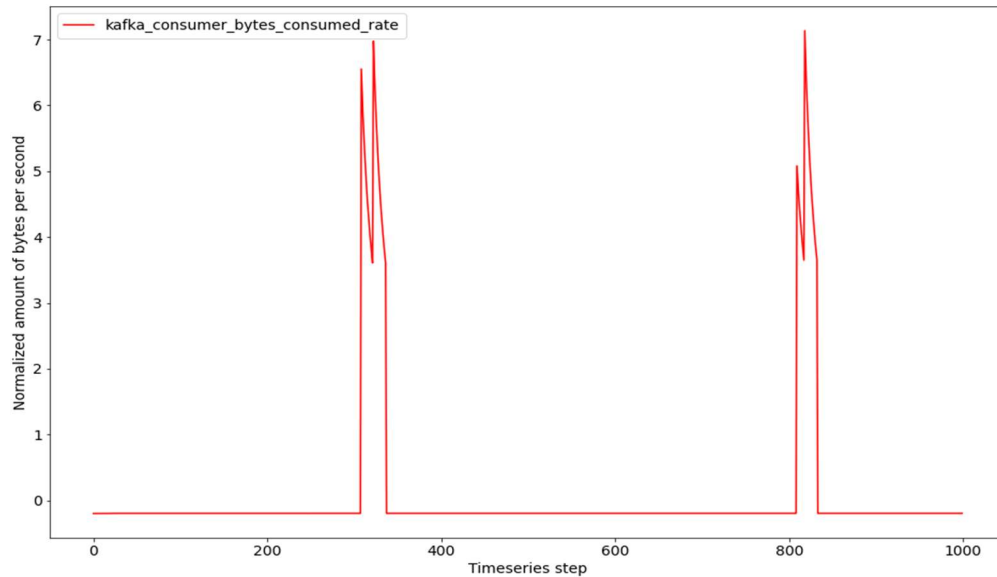


Figure 4: Normalized Data of the Kafka_Consumer_Bytes_Consumed_Rate Metric

The peaks of anomalies are clearly distinguished on the presented graphs. In real practice, such peaks are obtained during DoS and DDoS attacks. The existing dataset was divided into two parts: three quarters was a training fetch and one quarter – a testing fetch. This approach allows avoiding the problem of the cold start of the model since it takes significantly less time and computing resources to find the local minimum at each iteration.

As a basic real-time model, a modification of the Streaming Linear Regression model embodied into the Spark MLlib library was used, which is a

linear regression model that modifies its weights on each new piece of data received using the gradient descent method.

The modification of the model developed by the tori is the autoregression AR(p), where p is the order of autoregression, i.e. the number of terms in the model corresponding to the previous p values of the time series. Such a model approximates ARMA(p,q) [18] with some accuracy at certain values of p. In addition to autoregression, ARMA(p,q) model includes a moving average

model, where p is the order of autoregression, and q is the order of the moving average model.

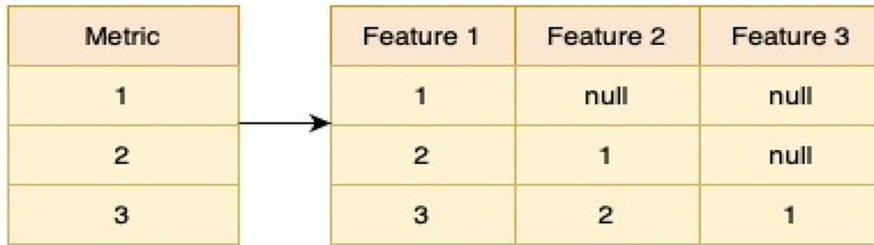
The technical implementation of the model was made in the form of a class added in the Scala programming language, inheriting from the Streaming Linear Regression class, and implementing a buffer of the previous p values of the

time series (Figure 5). Using this buffer, the time series is converted into a dataset with an attribute space consisting of values shifted relative to the current value of the time series (Figure 6). Thus, the problem is reduced to a regression problem, which can be addressed using methods, built into Streaming Linear Regression.

```
abstract class AR(p: Int) extends StreamingLinearRegressionWithSGD {
  var featuresHistoric: ListBuffer[Double] = ListBuffer.empty

  def addToFeatures(v: Double): ListBuffer[Double]
}
```

Figure 5: The Interface of the Autoregression Class



| Metric | Feature 1 | Feature 2 | Feature 3 |
|--------|-----------|-----------|-----------|
| 1 | 1 | null | null |
| 2 | 2 | 1 | null |
| 3 | 3 | 2 | 1 |

Figure 6: Transformation of a Time Series to a Dataset for Linear Regression

3. RESULTS

3.1 Anomaly Detection

A comparison was made between K-Means Streaming, which was initially given a dataset for training, and a model using K-Means to obtain a pre-trained model.

The number of clusters was assumed to be $K = 3$ by the elbow method.

The K-Means model, built into Spark MLlib Streaming operates according to the classical scheme: first, it is trained based on a training dataset in streaming mode, and then it can be applied to real data. The combined K-Means Streaming model, implemented by the authors, acts differently: first, it uses a pre-trained classical (rather than real-time) K-Means model, whose accuracy is higher due to additional model tuning and optimization of parameters that are not available in streaming mode. Then, the real-time Streaming K-Means model is used through the adapter (Figure 2), working with real data. This approach allows a more flexible configuration of model parameters.

As a result of the operation of the two described models, the centers of their clusters differ significantly, which is explained by a large number of iterations in the classical version of K-Means,

compared with the Spark MLlib Streaming K-Means model, in which the emphasis is made on the computing speed.

Let us consider the results of determining the coordinates of cluster centers for the standard K-Means model and the proposed Streaming K-Means model, based on two metrics that directly determine the anomaly of an event, as:

- 'message rate' – the number of messages per second;
- 'kafka_consumer_bytes_consumed_rate' – the number of bytes per second, received by the consumer.

For the model, obtained by the standard K-Means, the cluster centers are practically deterministic, since the number of iterations can be set during training, and when their number is higher than 10, the centers coincide with high accuracy.

For Streaming K-Means, the resulting centers of the model that initially have received the same data may differ significantly in position for different variants.

Let us consider two cases of training of Streaming K-Means models with obtaining the coordinates of the centers for three clusters, specified in two-dimensional space (Table 3).

Table 3: Examples of Resulting Cluster Centers

| Cluster No | Coordinates of the centers | |
|------------|--|---|
| | The first case | The second case |
| 1 | (7.074178082686644, -0.11795465590978567) | (8.840045919479378, 0.5943960022854669) |
| 2 | (0.6992464872049381, 5.296479067828162) | (-0.08328441646883958, -0.005599962336461511) |
| 3 | (-0.08334663515471895, -0.18317698019505607) | (-0.08328441646885959, -0.005599962336481511) |

It is seen from Table 3 that the coordinates of the cluster centers differ from each other. Accordingly, the accuracy of the model differs in different tests. On the test dataset, clustering models have the following quality indicators:

- K-Means Streaming with a model pre-trained by K-Means: TPR = TNR = PPV = NPV = 1.
- K-Means Streaming, which was first given a training fetch: TPR ∈ [0.90 ; 0.95], TNR = PPV = 1, NPV = 0.99, where TPR is sensitivity, TNR is specificity, PPV is accuracy, and PNV is the proportion of false omissions [14].

The decrease in quality indicators for the second model is associated with the non-determinism of cluster centers after learning of K-Means Streaming based on a test fetch.

3.2 Time Series Prediction

The following models were used for comparative analysis:

- modification of autoregression, considered in this article, with the order of autoregression p=5;
- classical autoregression, trained on a training dataset with the order p=5;
- integrated autoregression of ARIMA moving average model with autoregression order p=5, differentiation degree d=2, and moving average order q=1.

Figure 7 below shows a graph with the results of the predictions of the studied models.

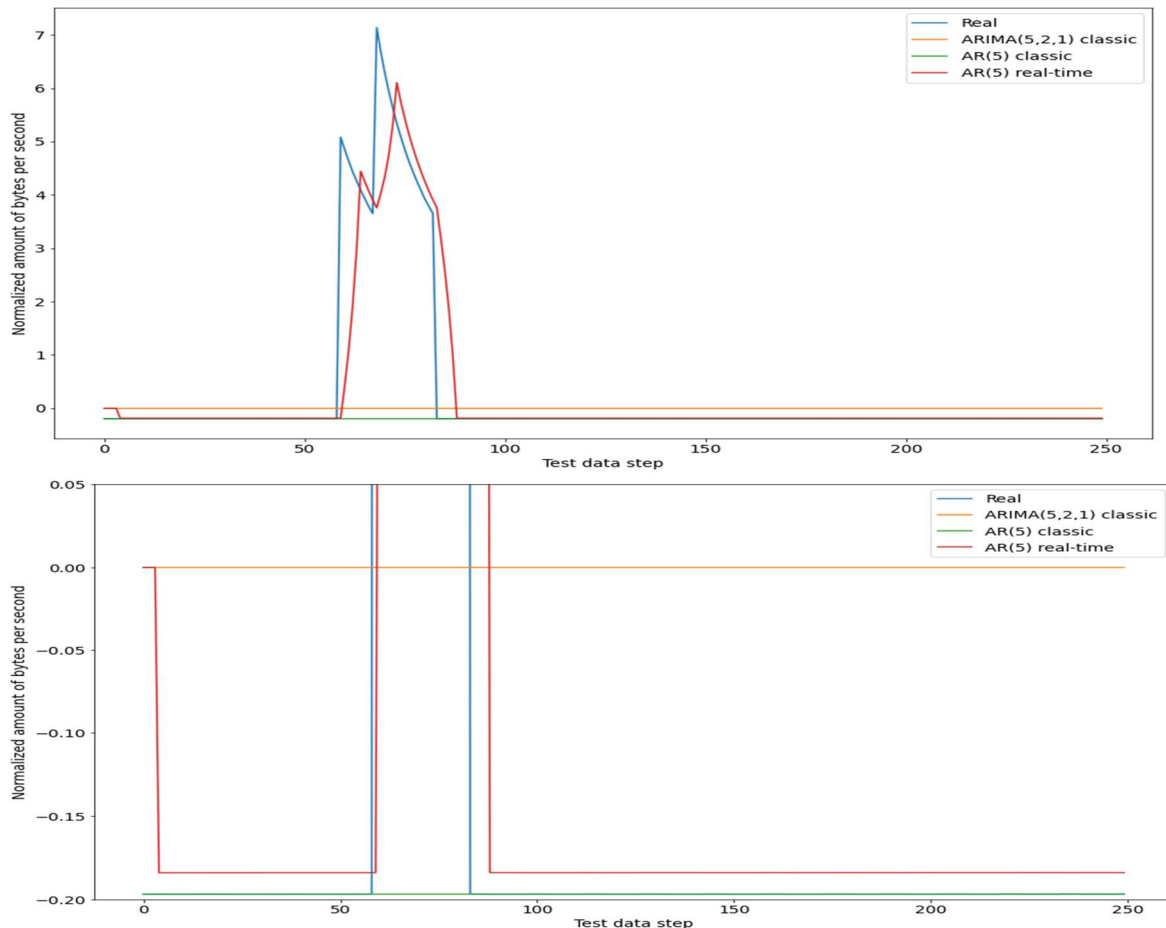


Figure 7: Results of Model Prediction on the Test Fetch

As can be seen from Figure 7, in the case of a dataset without periodicity, classical models cannot cope with the problem of predicting time series. Such behavior is easily explained and is since the anomalies are non-periodic, which means they do not fit into the pattern to which the classical model was adapted.

4. DISCUSSION

The considerable interest of the academic community in the problem of real-time processes monitoring in various fields of activity and, consequently, improving existing approaches to detecting anomalies and predicting time series values has recently been caused by the desire to use different starting points to solve these problems. There are several such starting points, namely, approaches based on classical models and algorithms, developing special real-time methods, and applying neural networks.

When solving the problem of detecting anomalies, the range of research is gradually shifting from using classical methods, pre-trained on finite samples [10, 19-21] or based on identifying statistical dependencies [11, 22, 23] and using only neural networks [14, 23, 24], to special real-time methods [25-28] and combinations of various methods [15, 20, 24].

For datasets of certain structures, classical methods, based on the calculation of the location and variance of the sample, obtained from relatively unknown zero-order statistics [16], or using a statistical lever (credit leveraging) – an information-theoretic statistical measure to identify outliers [23], give acceptable results.

In the light of the evolution of the requirements for the methods used towards streaming data processing and an increase in the number of highly dynamic real-time time series analyzed, various hybrid approaches combining real-time methods [27, 28] with settings in the form of neural networks are becoming increasingly interesting [14, 24]. When adding the requirement to work in distributed networks, only real-time methods remain in the focus of interest.

The proposed approach to solving the problem of detecting anomalies based on log-based metrics of technological platforms with rapidly changing metric values is based on the idea of implementing K-Means, built into Spark MLlib Streaming, by supplementing the initial markup function to implement the "hot start" model.

When developing special real-time methods, the emphasis is placed on the computation

speed in order to have time to produce these computations before receiving the next portion of streaming information. Therefore, the issue of bringing the values of clustering quality indicators to similar values of classical methods with a large allowable number of iterations remains and requires further research.

When solving the problem of predicting the behavior of time series, one can note a gradual shift in emphasis towards developing special real-time methods [14] based on using classical approaches [29] and neural networks [30]. However, this problem is not yet widely presented in publications.

5. CONCLUSION

The present article considers the use of ML when working with time series in real-time to solve problems of detecting anomalies and predicting values. The article presents an overview of available works and open source solutions in the field of online learning, as well as studies and implements modifications of classical AR(ARMA) and K-Means models for real-time operation, which allows not only parallelizing computations using the deferred computing paradigm but also configuring the model on-the-fly in the Apache Spark ecosystem.

To test the obtained algorithms, a dataset, consisting of JVM and Kafka metrics was prepared for the transmission of messages by two producers. When creating a dataset, characteristics, such as the number of messages per second, and the size of the transmitted messages were changed. Also, two anomalies were included in the dataset in the form of messages with a size many times larger than the average size of messages in non-anomalous sections. To detect anomalies and predict time series, the dataset was normalized. Static and correlating metrics were excluded. The `kafka_consumer_bytes_consumed_rate` metric was taken as a reference value. The dataset was divided into two parts: for training and testing.

In the problem of predicting time series for comparative study on a test dataset, AR(5) and the modification of Real-Time AR(5), proposed by the authors were pre-trained. The classical AR(5) model, as expected, failed to predict the anomalous section. Real-Time AR predicted both anomalous and non-anomalous sections; however, the accuracy of the model was worse than that of the classical version, since the number of iterations of Streaming K-Means was limited by the need to process streaming data at high speed. Improving the accuracy of real-time algorithms can be the subject of further research and improvements.

In the anomaly detection problem, the Real-Time K-Means modification, proposed by the authors, and the implementation of Streaming K-Means from the Apache Spark Mllib library, pre-trained based on a training dataset, were compared. When considering all non-correlating metrics, the cluster centers of the classical and online models appeared to be separated, however, when considering only metrics, directly related to the anomaly, the centers coincided with high accuracy.

This fact indicates the operability of the Real-Time K-Means model and the need for a "hot start of the model", because with a "cold start" the efficiency of the algorithm decreases. However, when changing the profile of metrics, for example, at an increase in the load on the service, the online model can easily adjust to the new data, unlike the classic version, which cannot adjust.

Summing up, the modifications of real-time models proposed by us for working with time series contain new results in terms of implementing the "hot start" method for real-time K-means, as well as add the ability to control the prediction horizon for Real-Time AR(5) when working with streaming data in a distributed environment. The conducted experiments indicate further research and development vector in progressing online learning models in the Apache Spark ecosystem.

REFERENCES:

- [1] P.J. Brockwell, and R.A. Davis, *Introduction to time series and forecasting*. Springer, 2016.
- [2] A. Aldweesh, A. Derhab, and A.Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues," *Knowledge-Based Systems*, Vol. 189, 2020, Art. No. 105124.
- [3] Y.G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Ait-Bachir, and V. Strijov, *Time series forecasting using RNNs: An extended attention mechanism to model periods and handle missing values*, March 29, 2017. [Online]. Available: <https://arxiv.org/pdf/1703.10089v1.pdf>
- [4] A. Sarvani, B. Venugopal, and N. Devarakonda, "Anomaly detection using k-means approach and outliers detection technique", in K. Ray, T. Sharma, S. Rawat, R. Saini, and A. Bandyopadhyay (Eds.), *Soft computing: Theories and applications. Advances in intelligent systems and computing*, Vol. 742. Singapore: Springer, 2019, pp. 375-385. https://doi.org/10.1007/978-981-13-0589-4_35
- [5] V. Lemaire, O. Alaoui Ismaili, A. Cornu'ejols, and D. Gay, "Predictive k-means with local models", in W. Lu, and K.Q. Zhu (Eds.), *Trends and applications in knowledge discovery and data mining. PAKDD 2020. Lecture notes in computer science*, Vol. 12237. Cham, Switzerland: Springer, 2020, pp. 91-103. https://doi.org/10.1007/978-3-030-60470-7_10.
- [6] T. Tsigkritis, G. Groumas, and M. Schneider, "On the use of k-NN in anomaly detection", *Journal of Information Security*, Vol. 9, 2018, pp. 70-84. <http://dx.doi.org/10.4236/jis.2018.91006>
- [7] *Unified engine for large-scale data analytics*. [Online]. Available: <https://spark.apache.org/> (access date: October 1, 2021).
- [8] *Apache Hadoop*. [Online]. Available: <https://hadoop.apache.org/> (access date: October 1, 2021).
- [9] V. Kozitsin, I. Katser, and D. Lakontsev, "Online forecasting and anomaly detection based on the ARIMA model", *Applied Sciences*, Vol. 11, No. 7, 2021, Art. No. 3194. <https://doi.org/10.3390/app11073194>
- [10] V. Zeufack, D. Kim, D. Seo, and A. Lee, "An unsupervised anomaly detection framework for detecting anomalies in real-time through network system's log files analysis", *High-Confidence Computing*, Vol. 1, No. 2, 2021, Art. No. 100030.
- [11] S. Chen, G. Jin, and X. Ma, "Detection and analysis of real-time anomalies in large-scale complex system", *Measurement*, Vol. 184, 2021, Art. No. 109929. <https://www.sciencedirect.com/science/article/abs/pii/S0263224121008666>
- [12] A.S. Alghawli, "Complex methods detect anomalies in real time based on time series analysis", *Alexandria Engineering Journal*, Vol. 61, No. 1, 2022, pp. 549-561. <https://reader.elsevier.com/reader/sd/pii/S1110016821003951?token=B823CED8598AAC7DCC9AC34673F2B52765413196DF858577AB726F3FC00A6C759F15EC76DB89AE6F2E47FA31C6195700&originRegion=eu-west-1&originCreation=20211109114841>
- [13] C.A. Bollmann, M. Tummala, and J.C. McEachen, "Resilient real-time network

- anomaly detection using novel non-parametric statistical tests”, *Computers & Security*, Vol. 102, 2021, Art. No. 102146. <https://doi.org/10.1016/j.cose.2020.102146>
- [14] Z. Wang, Y.H. Zhou, and G.M. Li, “Anomaly detection by using streaming k-means and batch k-means”, in *2020 5th IEEE International Conference on Big Data Analytics (IEEE ICBDA 2020)*, Xiamen, China, 8–11 May 2020, pp. 11–17.
- [15] *Clustering - RDD-based API*. [Online]. Available: <https://spark.apache.org/docs/latest/mllib-clustering.html> (access date: October 1, 2021).
- [16] R.H. Shumway, and D.S. Stoffer, *Time series analysis and its applications: With R examples*, 3rd ed. Springer, 2011, 609 p
- [17] E.J. Hannan, *Multiple time series. Wiley series in probability and mathematical statistics*. New York, NY: John Wiley & Sons Inc., 2009.
- [18] O. Anava, E. Hazan, Sh. Mannor, and O. Shamir, “Online learning for time series prediction”, *Proceedings of the 26th Annual Conference on Learning Theory, PMLR*, Vol. 30, 2013, pp. 172-184.
- [19] Md T.R. Laskar, J.X. Huang, V. Smetana, Ch. Stewart, K. Pouw, A. An, S. Chan, and L. Liu, “Extending isolation forest for anomaly detection in big data via k-means”, *ACM Transactions on Cyber-Physical Systems*, Vol. 5, No. 4, 2021, Art. No. 41. <https://doi.org/10.1145/3460976>
- [20] J. Henriques, F. Caldeira, T. Cruz, and P. Simoes, “Combining K-Means and XGBoost Models for Anomaly Detection Using Log Datasets”, *Electronics*, Vol. 9, No. 7, 2020, Art. No. 1164. <https://doi.org/10.3390/electronics9071164>
- [21] S. Ying, B. Wang, L. Wang, Q. Li, Y. Zhao, J. Shang, H. Huang, G. Cheng, Z. Yang, and J. Geng, “An Improved KNN-Based Efficient Log Anomaly Detection Method with Automatically Labeled Samples”, *ACM Transactions on Knowledge Discovery from Data*, Vol. 15, No. 3, 2021, Art. No. 34. <https://doi.org/10.1145/3441448>
- [22] T. Fawcett, “An introduction to ROC analysis”, *Pattern Recognition Letters*, Vol. 27, No. 8, 2006, pp. 861–874.
- [23] J. Ko, and M. Comuzzi, “Keeping our rivers clean: Information-theoretic online anomaly detection for streaming business process events”, *Information Systems*, Vol. 104, 2022, Art. No. 101894. <https://www.sciencedirect.com/science/article/abs/pii/S0306437921001125>
- [24] X. Zhang, J. Mu, X. Zhang, H. Liu, L. Zong, and Y. Li, “Deep anomaly detection with self-supervised learning and adversarial training”, *Pattern Recognition*, Vol. 121, 2022, Art. No. 108234. <https://www.sciencedirect.com/science/article/abs/pii/S0031320321004155>
- [25] V. Kuznetsov, and M. Mohri, “Time series prediction and online learning”, *29th Annual Conference on Learning Theory, PMLR*, Vol. 49, 2016, pp. 1190-1213.
- [26] D. Fotakis, Th. Lianas, G. Piliouras, and S. Skoulakis, “Efficient online learning of optimal rankings: Dimensionality reduction via gradient descent”, in *Advances in Neural Information Processing Systems 33: Proceedings of the international conference "Neural Information Processing Systems 2020" (NeurIPS 2020)*. Curran Associates, Inc., 2020.
- [27] *Streaming linear regression*. [Online]. Available: <https://spark.apache.org/docs/latest/mllib-linear-methods.html#streaming-linear-regression> (access date: October 1, 2021).
- [28] *Streaming k-means*. [Online]. Available: <https://spark.apache.org/docs/latest/mllib-clustering.html#streaming-k-means> (access date: October 1, 2021).
- [29] R.J. Hyndman, and G. Athanasopoulos, *Forecasting: Principles and practice*, 3rd ed. Melbourne, Australia: OTexts, 2021.
- [30] Z. Han, J. Zhao, H. Leung, K.F. Ma, and W. Wang, “A Review of Deep Learning Models for Time Series Prediction”, *IEEE Sensors Journal*, Vol. 21, No. 6, 2021, pp. 7833-7848. <https://ieeexplore.ieee.org/abstract/document/8742529>