Event stream classification with limited labeled data for e-commerce monitoring

Alexander Zimin
Intelligent Cloud Control
Amazon.com, Inc.
zimina@amazon.com

Igor Mishchenko
Intelligent Cloud Control
Amazon.com, Inc.
immishch@amazon.com

Rebecca Steinert
Intelligent Cloud Control
Amazon.com, Inc.
rsteinrt@amazon.com

***Abstract*—Monitoring and diagnostics of large software systems is crucial to ensure uninterrupted functioning of modern businesses. Reliability engineers have to rely on automatic event processing to identify and mitigate any potential disruptions of the system health from underlying computer networks. As obtaining impact labels for individual events is expensive, systems operators usually maintain only a small representative dataset, making it hard for machine learning practitioners to train models on large-scale data streams.**

By formulating the problem within the multiple instance learning framework, we propose an approach to event classification that can be effectively trained using this limited information. Our evaluation results show potential 65% reduction in minutes spent by the network reliability engineers on disruption investigations when the proposed model is used. By automatically quantifying the network impact, the proposed approach streamlines the investigation process and reduces the risk of unnecessary wake-up calls among on-call reliability engineers and resolver personnel.

***Index Terms*—Machine learning, Multiple instance learning, Event stream classification, Fault Management, Service Monitoring**

I. Introduction

Nowadays modern businesses build large software systems to support their digital business models. According to the "Microservices Adoption in 2020" report [1] around 77% of respondent organizations utilize micro service architecture including software-enabled companies like Netflix, Amazon and Spotify [2]. The widespread adoption of the microservice architecture is attributed to its feature flexibility, better scalability and faster delivery. However, this architecture comes with its own challenges. In addition to increased complexity in design and service management, 27% of respondents mention monitoring and observability as a major challenge in operating such systems [1].

These systems can easily grow to thousands of services with thousands of instances [3] making it non-trivial to find a root cause of the disruption whenever it occurs. Reliability engineers often rely on visualization tools and manual search through logs to identify the responsible service, find the reason of the disruption and perform the correct mitigation actions [4]. Since every minute of downtime is unacceptable loss of customer trust, the automation of each step of the process can allow the engineers to focus on more complex issues and decrease the overall time to problem resolution. This motivates the research focused on automatic localization of the disruption root causes in microservice architectures, e.g. see [5]–[7].

The root cause localization is additionally complicated when a disruption comes not from the software side but from the underlying networking infrastructure that supports communication between parts of the system. State of the art network monitoring solutions provide a stream of events that indicate potential problems in various parts of the network [8], but not every event affects the health of the software that uses the network. Being able to quickly identify cases of network failures affecting software reduces the time to localization and allows reliability engineers to focus on mitigation.

This problem is a specific case of a general event classification problem that also appears in other settings like intrusion detection [9], purchase intention [10] or surveillance [11]. However, in the context of computer networks it is presented with unique challenges. Not every network monitoring event results in the disruption of the service operations. In fact, the disruptions are much rarer: there can be a single disruption per week, while the monitoring system produces millions of events per week. This complicates the usage of the raw event stream during the localization process and makes it hard to clarify the effect of each individual event on the system health. The discrepancy between the number of events and disruptions also affects the granularity of the available labeled data. Typical machine learning solution would require a label for each event to train a classifier. To obtain a single label, an event has to be examined in combination with other systems and signals to determine the range of the impact and requires very

specific domain knowledge that also limits the amount of people who can perform such labelling. However, system operators usually maintain a dataset of the global system disruptions that contains information about the times of the disruptions and their root causes. Even though it can be very cost-effective to utilize this already labeled data to learn an impact classifier, this approach comes with the challenge that the disruption cannot be attributed to a single event but rather to a group of events that appeared during time leading to the disruption.

We propose to tackle the problem of event classification from limited labels with a multiple instance learning approach. The proposed approach aggregates the features of the events in a given context window to construct a single feature vector representing the overall network state at the current time and then uses the resulting embeddings to classify the impact. This way we can train the model using the dataset of global disruptions and determine the impact of the network on the system health without relying on individual events. Under operational conditions of a large e-commerce website, our approach contributes to reducing 65% of the investigative time spent by network reliability engineers during disruptions.

Our main contributions are: multiple instance learning based approach to event classification with limited labels, evaluation of the approach in terms of classification performance and business impact and details of the system implementation to operate the resulting model in an industrial environment.

The rest of the paper is structured as follows: Section II surveys the prior work on event classification. Section III introduces the problem setting, multiple-instance learning, our proposed approach and details its engineering implementation. Section IV presents the experimental setting and our evaluation results together with the ablation study. Finally, in Section V we discuss various aspects of the proposed approach like projected business impact, its general applicability and computational complexity.

II. Related work

The root cause localization in microservice architectures typically focuses on finding the service that caused a disruption. Such approaches, [5]–[7], rely on service metrics like CPU and memory utilization, API latencies, etc. to construct a dependency graph of the services and track the propagation paths of service failures. This works well whenever the disruption originates in the service logic and caused, for example, by a faulty deployment or insufficient scaling. However, it can be challenging to detect issues caused by the disruptions in the supporting computer network because there are no responsible services and the issues can manifest themselves in ways similar to regular disruptions, e.g. by increased API latencies or increased CPU utilization due to some hosts being unreachable.

In the field of computer networking there is a line of work that focuses on minimizing the disruptions to the traffic that goes through the network. In [12] the authors propose a framework for building and evaluating resilient network architectures that serves as a preventive measure for the traffic disruption. A simple passive algorithm to monitor service availability by counting the number of packets arriving to the designated hosts is presented in [13]. The focus of [14] is to improve the existing techniques for prompt analysis of underlying traffic patterns and develops a framework to discover interesting and unusual traffic flows that can be indicative of disruptions like Denial of Service attack. These and other similar techniques are used in network monitoring solutions [8] and their output is aggregated in the event stream that we use as an input in our approach.

A related research direction within the computer network domain is the intrusion detection. The solutions usually rely on the applications of anomaly detection [15]. For example, in [16] the authors build an autoencoder-based anomaly detection system applied to the log data collected across the network. In [17] identification of users' activities as normal or anomalous is approached by building an intrusion detection system using random forest classifier. Unsupervised language modeling techniques to detect attacks on computer network traffic data are used by [18]. The authors of [9] approach the problem as anomaly detection using Bayesian networks that allows to incorporate external signals like the system health to improve the intrusion detection accuracy. Even if some of the works attempt to use the service health information as input, they focus on the network health and do not expand to the implications of the detected attacks on the systems that run on these networks.

The general problem of event stream classification is studied across very different domains. In [19] the discrete event streams in social media are studied with the goal of clustering the events in online manner to categorize the traffic and to discover data shifts by detecting new clusters. In the predictive maintenance scenario [20], the goal is to either estimate the state of machine deterioration, or anticipate a specific type of failure, in order to prevent it from happening. The approach relies on autoencoder-based anomaly detection to detect anomalous events that are indicative of the potential failures. The implementation of the system for

event classification in the context of intrusion detection is detailed in [21]. The authors use an ensemble of multiple approaches like naïve Bayes classifier that operates on individual events, clustering-based anomaly detection techniques and rule-based detectors. In the context of audio streams [11], event detection is used with audio sensors data in surveillance and monitoring applications. The events are defined using 10 minutes of audio data and are modeled using Gaussian Mixture Models to perform the classification into vocal vs. non-vocal events, and into normal vs. excited events. The authors of [22] study event classification in the context of distributed system monitoring. When using the proposed approach, time series data is generated from the event stream based on pre-defined queries to the event processing systems and then anomaly detection methods are applied to the resulting time series to detect and explain event impact. Most of the referred solutions either use some form of anomaly detection that requires them to build the model of normal system behavior or have enough labeled data to build regular classification models. In contrast, our approach lies in-between these two scenarios where we can train an impact classifier from the available sparse labels without requiring per-event labels.

Data stream classification is a more common problem setting that assumes the availability of the labels for every single event and is usually approached by incremental algorithms [23]. For example, the authors of [24] introduce the notion of dynamic context window that prevents the models from overfitting to recent changes and uses the k-nearest neighbors classifier to perform the classification from these windows. Many other regular classification models can be adapted to be used for data stream classification like SVMs [25] or extreme learning machines [26]. In the context of e-commerce frameworks [10], the prediction of online shoppers' intent is framed as a data stream classification problem and is approached using naïve Bayes classifier, C4.5 decision tree and random forest. All these methods benefit vastly from labeled data that is lacking in the setting studied in this paper.

Multiple instance learning (MIL) is a well-studied framework within the machine learning community. Refer to [27] for a comprehensive review of this area. All approaches to MIL can be split into two groups: instance and bag level ones. The direct instance level approach defines a model for each instance, makes assumptions on how the instance scores are aggregated and uses it to backpropagate the gradients during the training, as shown in [28] for neural networks and in [29] for logistic regression. Bag level approaches sometimes rely on computing the distance between the bags directly, like


```
{
  "createdAt": "2020-05-23 03:22",
  "device": {
    "type": "router",
    "location": {
      "region": 1,
      "data_center": 3,
      "rack": 13
    },
    "os_version": 1.3.2,
    "model": "x_18"
  },
  "event_type": "device:unreachable",
  "status": "created"
}
```


Fig. 1: An example of a networking monitoring event.

for example in [30], [31], or on constructing embeddings for the bags and passing them to regular classifiers, e.g. [32], [33]. The approach in this paper falls under the latter category as it offers good scalability, both during training and inference, and is better suited for MIL problems with low number of positive instances in the bags.

As we can see, detecting the impact of the networking state over the supported systems remains a challenging problem in the industry. Overall, we observe that the use of multiple instance learning algorithms for solving event classification with limited label information is underrepresented in literature. To the best of our current knowledge, a practically applicable solution to this problem for large scale businesses has not been provided until now.

III. Event classification with limited labels

In this section we present the formal problem setting, introduce multiple instance learning framework and detail our approach to the problem together with its implementation.

A. Problem statement

Network monitoring event is a data structure that contains information about the potential networking issue. It indicates the type and location of the device that gave rise to the event and optionally some meta-information about the device. Events can be split into different types according to the type of the issue that they represent, e.g. the device is unreachable or a link between two devices is down. Events can be stateful going through stages like "created", "escalated" or "mitigated". If this is the case, whenever an event changes its status it appears in the stream again with the new status. An example of a networking monitoring

Fig. 2: Event stream classification as a multiple instance learning problem.

event can be found in Figure 1. We construct a feature representation of each event by using one-hot encodings for all discrete fields like device and event type and concatenating them together (one-hot encodings are vector representations in which all of the elements are 0, except for one, which has 1 as its value, where 1 represents a boolean specifying a category of the element). For an event e we denote the resulting feature vector by $f(e) \in \mathbb{R}^d$.

Networking monitoring solutions output a stream of events e_t that, after feature extraction, becomes a stream of feature vectors $f(e_t)$. The task is to produce a stream of labels $\hat{o}_t \in \{0, 1\}$ that indicates whenever there is an impact of the network on the software system using it. At time t the classifier is naturally restricted to use events in the stream up to time t.

To train and evaluate the model we can access the disruptions database that contains timestamps of the disruptions and their root causes. All disruptions with a networking root cause are labeled as positive and the rest as negative. Formally the disruptions database is a set of tuples (τ_i, o_i, d_i), where τ_i is a timestamp of the i-th disruption, o_i is a label and d_i is the duration of the disruption that serves as an indicator of its severity.

B. Multiple instance learning

Multiple instance learning (MIL) is a machine learning problem, where the learner does not have access to labels at the data point level, but rather has access to group (bag) labels. Event classification can be formulated as MIL problem if we consider each individual event as a separate sample and all events occurring before the same disruption as belonging to a single bag, see Figure 2 for an example. There are several variants of MIL that make different assumptions about the number of positive samples in a bag required for the bag to be positively labeled [34]. In the context of computer networks even a single event can cause a disruption, therefore it is natural to consider the bag to have a positive label if at least one event is positive.

There are several ways to approach the training of the model in multiple instance learning. In general, all algorithms for MIL can be split into instance level and bag level ones. Instance level algorithms train models that produce scores or labels for each instance (event)

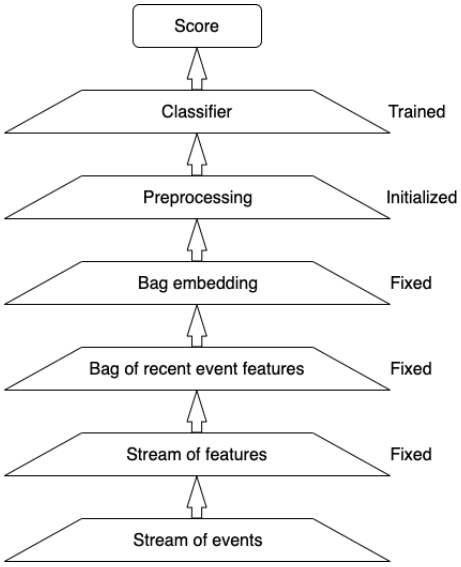

Fig. 3: The high level architecture of our approach. Event features are created using one-hot encodings, bags are combined from the events in context windows of a fixed size, bag embeddings are computed using the summation over the event features, preprocessing steps (scaling and dimensionality reduction) are initialized from the bag embeddings on the training dataset and the classifier is trained using the pre-processed embeddings as inputs and labels from the training dataset.

and then use the assumption on the bag structure to aggregate labels to the bag level. They typically work well only when the proportion of the positive instances in the positive bags, so called witness ratio, is high [27]. For problems with low witness ratio, bag-level algorithms that operate directly on the bags tend to perform better with a downside that they are not able to produce instance-level scores.

C. Our approach

The high level architecture of our approach based on MIL is detailed in Figure 3. For each disruption we use its timestamp to construct the corresponding bag by gathering events that appeared in the stream prior to the disruption: for any disruption (τ_i, o_i, d_i) we define a bag $B_{\tau_i} = \{e_t | \tau_i - h \leq t \leq \tau_i\}$ with h being the length of the context window. For a bag B we compute its embedding $g(B) \in R^d$ as a sum of the event feature vectors: $g(B) = \sum_{e \in B} f(e)$. Since the event features are constructed using one-hot encodings, the bag embeddings represent the counts of each discrete value for various event dimensions, e.g. the number of events per different device types or the number of different event types. Before being fed to the classifier,

the bag embeddings are scaled to $[0, 1]$ using min-max scaler initialized on the training dataset and are optionally passed through a dimensionality reduction pipeline using Principal Component Analysis (PCA) [35]. Whether the dimensionality reduction is used or not is a hyper parameter that is optimized during evaluation.

The resulting embeddings are passed to a binary classifier that outputs a probabilistic score representing the likelihood of the impact. Note that only the timestamp of the disruption is used, so the score can be produced for any given time point. This can be useful, for example, in a pro-active setting where the model predicts potential impact before a system-wide disruption occurs.

In some situation it can be beneficial to train the classifier with a weighted loss, for example to fix class imbalance in the training dataset or to align the loss function with the business metrics. For example, we can guide the model to focus more on the longest disruptions by defining a weight for the disruption (τ_i, o_i, d_i) as $w_i = \sigma(d_i - D)$, where σ is a sigmoid function and D is a breakpoint duration. The intention is to have disruptions that last exactly D minutes to have a weight of 0.5 and use sigmoid smoothing for longer and shorter ones.

During inference a label $\hat{o}_t$ for time t is produced by constructing the appropriate bag B_t, computing its embedding $g(B_t)$ and applying the trained classifier.

D. Engineering implementation

In this section we describe the implementation of the system that can operate the model architecture described in Figure 3 in a real industrial setting.

The main challenge in running the resulting model is the variability in the number of events in the output from the network monitoring solutions. This number can fluctuate between several dozens to thousands per minute and can be even larger during peak business hours. It is important for the system to be able to autoscale to meet such sudden increases in traffic while maintaining reasonable operational costs.

Our solution is to utilize the built-in scalability of AWS Lambda and SageMaker [1]. The high-level diagram of the system is shown in Figure 4. It can be split into three main parts: StreamUpdater, ModelInvoker and SageMaker endpoint.

The role of StreamUpdater is to populate the database, Amazon DynamoDB in this case, with events. The network monitoring system pushes updates through Amazon SNS that triggers the StreamUpdater deployed as an AWS Lambda function.

ModelInvoker performs three main tasks:

- collecting most recent events from the database,

[1]More information on services' features can be found on https://aws.amazon.com/lambda/ and https://aws.amazon.com/sagemaker/

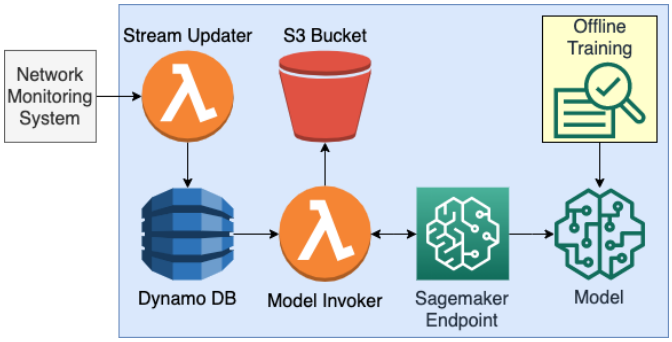

Fig. 4: Diagram of the system that implements the architecture from Figure 3.

- invoking the endpoint (Amazon SageMaker) hosting the model,
- saving the responses in an Amazon S3 bucket.

It is also implemented as an AWS Lambda function, however, it is not triggered on every single event. Since there is a delay between the start of the disruption and when the engineers look at the system output, it is enough to produce the output once per minute. This helps to minimize the costs by significantly reducing the amount of AWS Lambda calls, without any impact for the end consumers. In case the output needs to be consumed at a finer granularity, e.g. by an automated mitigation system, the frequency of calls can be easily increased to the desired rate. As discussed in Section III-A the event stream can contain multiple versions of the same event with different statuses. ModelInvoker ensures that only the most recent version of each event is kept and the rest are filtered out. Thus, given that the StreamUpdater maintains an up-to-date state of the stream, we are able to collect the events, filter them and aggregate into a single bag. The resulting list of events is then passed to the endpoint that provides a score for every request, which is then put into an S3 bucket, where it can be accessed by any UI and tooling built by the reliability engineers.

The classifier is trained offline and deployed to the SageMaker endpoint that takes care of scaling and monitoring. In case the number of requests is expected to be limited, even single-instance solutions without autoscaling can be viable. For example, in our test environment, single AWS EC2 t2.medium instance was sufficient to process requests with maximal number of events maintaining the average latency of 0.41 seconds with the 99th percentile of 2.12 seconds.

IV. Evaluation results

In this section we discuss the evaluation results of the proposed approach and compare its performance to a number of baselines. We discuss the choice of

TABLE I: THE EVALUATION RESULTS ON THE DISRUPTION DATABASE. THE NUMBERS SHOWN ARE THE MEAN ROC AUC SCORES COMPUTED BY A 5-FOLD CROSS-VALIDATION WITH THE CORRESPONDING STANDARD DEVIATIONS.

Level	Method	Context: 15 min	Context: 10 min	Context: 5 min
Instance	SI-LR	0.40 ± 0.18	0.46 ± 0.17	0.54 ± 0.09
	SI-LR-W	0.58 ± 0.15	0.54 ± 0.18	0.60 ± 0.06
	SI-RC	0.44 ± 0.10	0.43 ± 0.15	0.51 ± 0.15
	SI-RC-W	0.56 ± 0.20	0.54 ± 0.13	0.63 ± 0.06
	SI-SVM	0.46 ± 0.07	0.34 ± 0.14	0.42 ± 0.11
	SI-SVM-W	0.48 ± 0.12	0.58 ± 0.11	0.55 ± 0.11
	MIL-SVM	0.58 ± 0.13	0.51 ± 0.02	0.51 ± 0.09
	MIL-SVM-W	0.55 ± 0.08	0.55 ± 0.13	0.57 ± 0.05
Bag	STK	0.54 ± 0.08	0.56 ± 0.06	0.48 ± 0.14
	Embed-LR	0.71 ± 0.04	0.72 ± 0.07	0.72 ± 0.05
	Embed-RC	0.71 ± 0.03	0.71 ± 0.07	0.73 ± 0.06
	Embed-SVM	**0.79 ± 0.04**	**0.81 ± 0.05**	**0.76 ± 0.04**

the threshold depending on the target usage of the model. In addition, we perform an ablation study that tests the influence of different components on the final performance.

A. Studied use case

We study a particular use case when the model is used during disruption investigations by the reliability engineers. In this situation the model output is used to guide the engineers for whom it takes roughly 5 minutes to respond to a call after the disruption is detected and start the investigation. The model output for shorter disruptions will not be relevant in this case and it is more relevant to choose the model that performs well on longer disruptions. For this reason we focus the evaluation on the disruptions with the duration of more than 5 minutes. It is still beneficial to use the whole dataset for the training so that the model can learn to distinguish the situations that have no impact at all versus situations that cause short impact. Note that this potentially creates a discrepancy between the training and test distributions that the algorithms need to take into account. In our approach we use the weighted loss function as described in Section III-C.

B. Evaluation setup

We use a dataset collected in a test environment mimicking operations of a large e-commerce system. In this dataset the witness ratio is very low (less than 1%) and for the studied use case we are mostly interested in the detecting the impact of the network. These conditions fit a bag-level MIL approach, so the method presented in this paper is expected to perform better than instance-level ones. We still use a number of instance-level algorithms as baselines during the evaluation to confirm this hypothesis.

The majority of disruptions are rather short, therefore if we do a typical train-validation-test split, we will get very few long disruptions in the test set and our test scores will be not be truly representative of the business value in the studied use case. Our solution is to employ cross-validation, so that in the end any model is evaluated on the whole dataset, and we settled on a 5-fold cross-validation strategy. Since the scores produced by the model are used directly by reliability engineers, we need a threshold agnostic way to evaluate model performances. We resort to using area under receiver operating characteristic curve (ROC AUC) due to its simplicity and probabilistic interpretation.

To summarize, for each algorithm tested in this paper, we perform 5-fold cross-validation: for each fold we train the model on all disruptions and compute its out-of-sample area under curve score on disruption that lasted longer than 5 minutes. We then report the averaged ROC AUC over the folds as the final score together with its standard deviation.

C. Hyperparameters

We run the proposed method from Section III-B with different regular classifiers to test their influence on the performance. We present results for three options that showed the best performance out of various classifiers that we have tested and that showcase difference between linear and non-linear approaches: logistic regression, ridge classifier and SVM (linear and RBF kernel versions) with regularization constant varying for all classifiers in [0.01, 0.1, 1, 10] range. We denote these versions as Embed-LR, Embed-RC and Embed-SVM respectively. For each classifier we have used a corresponding open source implementation in SciKit-Learn [36]. We train each version with a weighted loss functions with weights from Section III-C with D set to 5. For each classifier we run the method with and without applying dimensionality reduction by PCA. Whenever it is applied, the target dimension takes values in [5, 10, 15, 20]. The size of the context window, h, is another

hyper parameter and we compare the results for 5, 10 and 15 minutes.

D. Baselines

Our goal with these experiments is to assess the benefits of using multiple instance learning for event stream classification with limited labels and compare it to a regular supervised learning approach. Additionally, we present two exemplar MIL methods that operate on instance and bag levels.

a) Single instance classifiers: This basic baseline ignores the multiple instance learning structure of the dataset by transferring the labels from disruptions to events appearing prior to the disruption and then by training a classifier that takes a single event as an input and predicts its impact. We evaluate the same classifiers that are used in our approach: logistic regression, ridge classifier and SVM with regularization constant varying for all classifiers in [0.01, 0.1, 1, 10]. We denote these versions as SI-LR, SI-RC and SI-SVM respectively. Here we use the same open source implementation from SciKit-Learn that is used in our method. For a fair comparison we also train these methods with the weighted loss with the same weights used in our approach, see Section III-C, and denote the corresponding variants as SI-LR-W, SI-RC-W and SI-SVM-W.

b) Single instance with aggregation: This is a direct adaptation of regular supervised learning algorithms to MIL: it trains a model that takes a single event, outputs the probabilities of the impact and then aggregates the probabilities over the bags by taking the maximum value to get the final score for the bag [28]. In order to keep this similar to other baselines we use SVM as the base model and train it using backpropagation. We vary the regularization constant in the same range [0.01, 0.1, 1, 10] as for other methods. Effectively, at each step this method either decreases or reinforces the score of the event(s) that attained the maximum score in their bags. We denote it as MIL-SVM and the version trained with a weighted loss as MIL-SVM-W. This method is implemented using SGDClassifier[2] from SciKit-Learn with hinge loss.

c) Statistics kernel SVM: There are other approaches to bag-level MIL. Not every such method scales well with respect to the size of the bags, therefore we chose a bag-level comparator with favorable computation complexity, SVM with a statistics kernel (STK) from [37], and evaluated it using our dataset. This is a statistic kernel based SVM that defines a kernel that can be computed on the bags directly. We train it with the linear kernel and regularization constant taking values in [0.01, 0.1, 1, 10]. We have used the implementation of STK provided in MISVM library [3].

E. Classification results

The results of the evaluation for all three context windows lengths are shown in Table I. The best overall result of 0.81 ROC AUC is achieved by the Embed-SVM method with the window length of 10 minutes and this algorithm also has the best performance for other lengths as well. In comparison, the best ROC AUC score achieved by the single instance classifiers is 0.63, showing a clear benefit of approaching the problem setting within a multiple instance learning framework. All Embed methods have also shown more robust performance with the standard deviations being consistently smaller than for other methods. For this use case and the data set used, a window length of 10 minutes for the Embed-SVM method provides the optimal trade-off between the size of the bags and the quality of the signal. The drop from 0.81 to 0.76 between 10 and 5 minute windows can be explained by the fact that the shorter context window misses the events that actually caused the impact because of the delay between the problem appearing in the network and the impact being detected. On the same note, single instance methods generally performed better with a smaller context window that can be explained by potentially better witness ratio in this case even if the positive events can be missed out. Due to our focus on disruptions that lasted for more than 5 minutes, instance level methods that incorporated weights during training perform better than the corresponding regular versions.

F. Threshold choice

ROC AUC is a useful metric to compare the overall performance of the classifiers across all possible decision thresholds and is well fitted for the studied use case. However, when deploying the model in other use cases, it is important to decide on the right threshold, so in this section we wanted to discuss the corresponding trade-offs on the exemplar model from the performed evaluation.

Figure 5 shows the ROC curve of the best classifier from the evaluation, Embed-SVM with context window of 10 minutes. Depending on how the model output is used, it can be beneficial to choose the threshold that supports the corresponding business use case.

Without any additional constraints, the most optimal trade-off between the true positive and false positive rates is provided by threshold of 0.58 that corresponds

[2]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[3]https://github.com/garydoranjr/misvm

TABLE II: THE RESULTS OF THE ABLATION STUDY OF Embed-LR, Embed-RC AND Embed-SVM. THE NUMBERS SHOWN ARE THE MEAN ROC AUC SCORES COMPUTED By A 5-FOLD CROSS-VALIDATION WITH THE CORRESPONDING STANDARD DEVIATIONS.

Classifier	PCA	Weights	Context: 15 min	Context: 10 min	Context: 5 min
LR	✗	✗	0.58 ± 0.13	0.56 ± 0.15	0.60 ± 0.11
	✗	✓	0.67 ± 0.08	0.67 ± 0.10	0.72 ± 0.08
	✓	✗	0.59 ± 0.13	0.59 ± 0.16	0.58 ± 0.14
	✓	✓	0.71 ± 0.04	0.71 ± 0.07	0.72 ± 0.05
RC	✗	✗	0.57 ± 0.14	0.57 ± 0.09	0.59 ± 0.14
	✗	✓	0.69 ± 0.08	0.65 ± 0.10	0.72 ± 0.07
	✓	✗	0.58 ± 0.14	0.59 ± 0.15	0.57 ± 0.14
	✓	✓	0.71 ± 0.03	0.71 ± 0.07	0.73 ± 0.06
SVM	✗	✗	0.58 ± 0.14	0.63 ± 0.12	0.61 ± 0.13
	✗	✓	0.75 ± 0.05	0.73 ± 0.09	0.76 ± 0.04
	✓	✗	0.66 ± 0.12	0.71 ± 0.11	0.68 ± 0.12
	✓	✓	0.79 ± 0.04	0.81 ± 0.05	0.76 ± 0.02

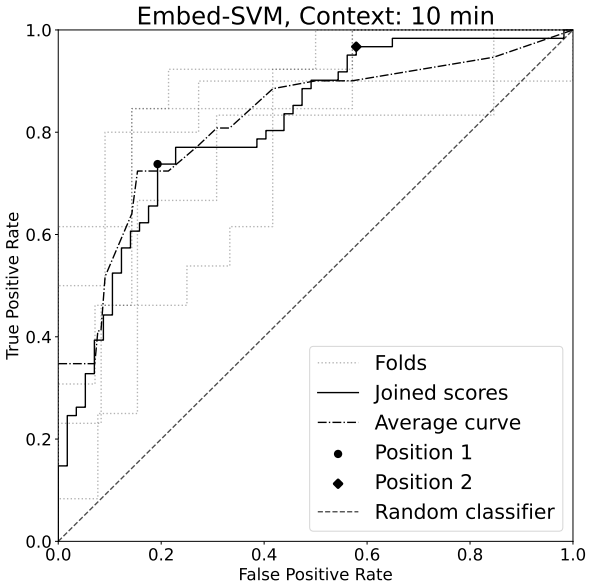

Fig. 5: The ROC AUC curve the Embed-SVM with context window of 10 minutes. Curves labeled as "Folds" are the individual ROC AUC curves for each of the 5 folds in the cross validation split. "Average curve" is the average over these 5 folds. "Joined scores" curve is computed by joining all scores from different folds together.

to Position 1 in Figure 5. This threshold guarantees around 20% of false positive rate together with 75% of true positive rate. The binary output produced with this threshold can be used in the system that operates preemptively, meaning that the high probability of the impact starts an investigation with the team of reliability engineers. The system will benefit from the low false positive rate because every false positive is expensive in terms of money and decreases human trust. In real life scenario the number of false positives can be driven further down by using extra monitoring signals from other systems, for example from business metrics.

Another use case is the usage of the model during the investigation of the ongoing disruption. In such situation it can be better to have fewer false negatives because the model can often be the single source of high level information about the network state and it can be very costly to overlook the real root cause of the ongoing disruption. This obviously comes at a price of higher false positive rate that can be tolerated. In case of Figure 5, we can choose threshold of 0.14 (Position 2) that guarantees the true positive rate of 95% at a cost of 60% false positive rate.

In the end, there is no single best threshold that works for every scenario. Each particular usage of the model has to be carefully studied to understand the implications of the model output and the corresponding costs.

G. Business impact

In practice it can be hard to translate the classification performance of a given model into its potential business impact. In order to evaluate how the proposed approach affects the disruption investigation process of a large e-commerce website, we tested the presented system on the use case of reducing the number of reliability engineers that are involved during the investigation. Whenever a disruption occurs, there is a number of reliability teams that are involved with each team focusing on a particular part of the system. All of them perform their own targeted investigations to cover all potential root causes and speed up the overall investigation process. One of these teams focuses on issues that arise from the problems with the computer network that supports the system. By looking at the historical disruptions we can estimate the potential

impact the model could have achieved if it was allowed to decide whether the networking reliability team should be involved in investigations of the current disruption, thus saving the networking team from unnecessary calls to non-networking disruptions.

During the studied period the correctly classified non-network related disruptions accounted for 65% of minutes spent by network reliability engineers on disruption investigations. At the same time the model made no false negative misclassifications, thus it was able to save these 65% of minutes without downsides. This affects not only the reliability engineering team, but also helps to streamline the investigation process by removing one potential root cause from the very beginning thus shortening the time to root cause localization and the overall website downtime.

H. Ablation study

The results presented in the Table I show the impact of using different classifiers in Embed methods. In this section we highlight the impact of the dimensionality reduction and the use of weighted objective on the achieved results. Table II shows the performance of the Embed-LR, Embed-RC and Embed-SVM when run with and without dimensionality reduction and the weighted loss function. Similarly to single instance methods, using weighted objective provides large benefit due to the evaluation focus on the longer disruptions.

In all cases, except for Embed-LR and Embed-RC with 5 minute context window, we observe the improvement in performance when the dimensionality reduction is performed, however, the weighted loss seem to be the bigger contributor to the performance in most cases. The noticeable exception is Embed-SVM when both components contribute more or less equally, especially in the case of 10 minute context window that provided the best overall performance.

V. Discussion

A. Other aspects of business impact

In Section IV-G we studied the business impact of the proposed approach in terms of potential minutes saved for the reliability engineering teams, but the potential model impact goes beyond that. Even though the model helps to confirm or eliminate only a single root cause of the disruptions, it automatically helps to decrease the system downtime in both cases, regardless whether the disruption is network-related or not. For large scale e-commerce businesses reducing the downtime of each disruption even by a single minute saves the company millions of dollars in revenue. Quantifying such an impact precisely can be challenging as it requires knowledge about dependency relationships (or causality models). Nonetheless, our evaluation results indicate that automated classification at the early stage of the fault management process is important for the operation of large scale businesses.

B. General applicability

The results presented in this paper support the usage of MIL-based approaches for event classification with limited information. However, the performance of the particular MIL method presented in this paper may not transfer directly to other datasets due to potential difference in the statistical properties of the data, and would hence likely require adaptation to the data domain and use case at hand. The network monitoring solution used influences a lot of implementation details as it determines the amount of information available for each event and their frequency. We would advise the practitioners who want to apply the proposed approach to experiment with other MIL methods as well, as they might be better suited for their particular problem domain, especially in cases with higher witness ratio where instance-level algorithms are likely to perform better. The main components of our approach, like the use of dimensionality reduction and weighted loss, are more likely to transfer between datasets as they are less dependent on the data source. Other factors of model productionization, such as determining the retraining frequency to keep the model up to date, is a matter of application domain and data properties.

C. Complexity and running time

The inference time complexity of the proposed approach is linear with respect to the size of the bags and feature vectors. In our experience the major bottleneck in the system is the feature extraction from events. This process requires parsing of the data structure and might become rather time consuming depending on the complexity of the event data structure. This is resolved by using more efficient parsers that can cache intermediate results so that each event is processed in a single pass.

VI. Conclusion

Making an efficient use of available datasets is a key factor to enable the application of machine learning in an industrial setting. In this paper we presented an approach to event stream classification with limited labels based on multiple instance learning framework that allows to classify the impact of events in the stream without having access to individual labels. We outlined the implementation of the approach that is scalable with respect to the frequency of the events in the stream and quantified the performance of the proposed algorithm

in terms of its classification performance and its business impact. By reducing the time spent by network reliability engineers on disruptions investigations by 65% and automatically confirming or ruling out one potential root cause of a disruption, our approach allows for smoother and faster root cause resolution process. This is especially valuable for large scale e-commerce businesses where reducing the downtime even by a minute on each disruption saves the company millions of dollars over time.

References

[1] O'Reilly. (2020) Microservices adoption in 2020. https://www.oreilly.com/radar/microservices-adoption-in-2020/. [Online; accessed 2021-07-15].

[2] microservices.io. (2020) Who is using microservices? https://microservices.io/articles/whoisusingmicroservices.html. [Online; accessed 2021-07-15].

[3] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, 2018.

[4] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[5] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 246–258.

[6] F. Lin, K. Muzumdar, N. P. Laptev, M.-V. Curelea, S. Lee, and S. Sankar, "Fast dimensional analysis for root cause investigation in a large-scale service environment," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–23, 2020.

[7] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10.

[8] S. Lee, K. Levanti, and H. S. Kim, "Network monitoring: Present and future," *Computer Networks*, vol. 65, pp. 84–98, 2014.

[9] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2003, pp. 14–23.

[10] K. Baati and M. Mohsil, "Real-time prediction of online shoppers' purchasing intention using random forest," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2020, pp. 43–51.

[11] P. K. Atrey, N. C. Maddage, and M. S. Kankanhalli, "Audio based event detection for multimedia surveillance," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 5. IEEE, 2006, pp. V–V.

[12] H. K. Nguyen, Y. Zhang, Z. Chang, and Z. Han, "Parallel and distributed resource allocation with minimum traffic disruption for network virtualization," *IEEE Transactions on Communications*, vol. 65, no. 3, pp. 1162–1175, 2017.

[13] I. Prieto Suárez, M. Izal Azcárate, E. Magaña Lizarrondo, and D. Morató Osés, "Detecting disruption periods on tcp servers with passive packet traffic analysis," in *SOFTENG 2015, The First International Conference on Advances and Trends in Software Engineering*. IARIA, 2015.

[14] M. Ahmed and A. N. Mahmood, "Network traffic analysis based on collective anomaly detection," in *2014 9th IEEE Conference on Industrial Electronics and Applications*. IEEE, 2014, pp. 1141–1146.

[15] Y. Shen, "The application of artificial intelligence in computer network technology in the era of big data," in *2021 International Conference on Computer Technology and Media Convergence Design (CTMCD)*. IEEE, 2021, pp. 173–177.

[16] Y. Dong, R. Wang, and J. He, "Real-time network intrusion detection system based on deep learning," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2019, pp. 1–4.

[17] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, pp. 213–217, 2016.

[18] B. J. Radford, B. D. Richardson, and S. E. Davis, "Sequence aggregation rules for anomaly detection in computer network traffic," *arXiv preprint arXiv:1805.03735*, 2018.

[19] T. Reuter and P. Cimiano, "Event-based classification of social media streams," in *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, 2012, pp. 1–8.

[20] S. Kauschke, M. Mühlhäuser, and J. Fürnkranz, "Towards semi-supervised classification of event streams via denoising autoencoders," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 131–136.

[21] V. Kumaran, "Event stream database based architecture to detect network intrusion: (industry article)," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013, pp. 241–248.

[22] H. Zhang, Y. Diao, and A. Meliou, "Exstream: Explaining anomalies in event stream monitoring," in *Proceedings of the 20th international conference on extending database technology (EDBT)*, 2017.

[23] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, 2018.

[24] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Proceedings of the 28th annual ACM symposium on applied computing*, 2013, pp. 801–806.

[25] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances in neural information processing systems*, pp. 409–415, 2001.

[26] R. Yang, S. Xu, and L. Feng, "An ensemble extreme learning machine for data stream classification," *Algorithms*, vol. 11, no. 7, p. 107, 2018.

[27] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon, "Multiple instance learning: A survey of problem characteristics and applications," *Pattern Recognition*, vol. 77, pp. 329–353, 2018.

[28] J. Ramon and L. De Raedt, "Multi instance neural networks," in *Proceedings of the ICML-2000 workshop on attribute-value and relational learning*, 2000, pp. 53–60.

[29] V. C. Raykar, B. Krishnapuram, J. Bi, M. Dundar, and R. B. Rao, "Bayesian multiple instance learning: automatic feature selection and inductive transfer," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 808–815.

[30] V. Cheplygina, D. M. Tax, and M. Loog, "Dissimilarity-based ensembles for multiple instance learning," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 6, pp. 1379–1391, 2015.

[31] J. Wang and J.-D. Zucker, "Solving multiple-instance problem: A lazy learning approach," 2000.

[32] X. Wang, Y. Yan, P. Tang, X. Bai, and W. Liu, "Revisiting multiple instance neural networks," *Pattern Recognition*, vol. 74, pp. 15–24, 2018.

[33] Y. Chen, J. Bi, and J. Z. Wang, "Miles: Multiple-instance learning via embedded instance selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 1931–1947, 2006.

[34] N. Weidmann, E. Frank, and B. Pfahringer, "A two-level learning method for generalized multi-instance problems," in *European Conference on Machine Learning*. Springer, 2003, pp. 468–479.

[35] K. Pearson, “LIII. On lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola, “Multi-instance kernels,” in *ICML*, vol. 2, no. 3, 2002, p. 7.