MACHINE LEARNING ON EVENT STREAMS IN A DISTRIBUTED ENVIRONMENT BY A STREAMLEARNER

Balaji T[1], Mohamed Yousuff A R[2], Abdul Naseer M[3] , Nandakumar P[4] , Fathima Begum M[5]
[1,2,3,4,5]Assistant Professor, Computer Science and Engineering
[1,2,3,4,5] C.Abdul Hakeem College Of Engineering And Technology

Abstract
Today, gigantic measures of gushing information from keen gadgets should be investigated consequently to understand the Internet of Things. The Complex Event Processing (CEP) worldview guarantees low-dormancy design discovery on occasion streams. Be that as it may, CEP frameworks should be stretched out with Machine Learning (ML) capacities, for example, internet preparing and derivation so as to have the capacity to identify fluffy examples (e.g. exceptions) and to enhance design acknowledgment exactness amid runtime utilizing incremental model preparing. In this paper, we propose a disseminated CEP framework indicated as Stream Learner for ML-empowered complex occasion discovery. The proposed programming model and information parallel framework design empower an extensive variety of genuine applications and take into account progressively scaling up and out framework assets for low-inactivity, high-throughput occasion master cessing. We demonstrate that the DEBS Grand Challenge 2017 contextual investigation (i.e., oddity discovery in savvy processing plants) coordinates consistently into the StreamLearner API. Our tests confirm versatility and high occasion throughput of Stream Learner.

Keywords: Complex Event Processing, Machine Learning, Stream Processing

1 INTRODUCTION AND BACKGROUND

As of late, the surge of Big Streaming Data being accessible from sensors [12], interpersonal organizations [17], and brilliant urban communities [3], has prompted a move of ideal models in information investigation all through all controls. Rather than bunch arranged handling [8, 14, 15], stream-situated information examination [7] is turning into the highest quality level. This has prompted the advancement of versatile stream preparing frameworks that execute the social inquiry model of social information base administration frameworks (RDBMS) as nonstop questions on occasion streams [2], and Complex Event Processing systems that implement pattern matching on event streams [6]. Inquiry driven stream handling, in any case, requests a space master to indicate the examination rationale in a deterministic question language with an inquiry that precisely de nes which input occasions are changed into which yield occasions by an administrator.

In any case, an unequivocal speci cation isn't generally conceivable, as the space master may rather be keen on a more dynamic question, for example, "Re-port me all inconsistencies that embellishment machine 42 encounters on the shop oor." In this case, it is infeasible to expressly determine all occasion designs that can be viewed as a peculiarity.

There have been di erent recommendations how to manage this issue. EP-SPARQL utilizes foundation ontologies to enable (com-plex) occasion handling frameworks with stream thinking [1] – while concentrating on the SPARQL question dialect. Then again, sev-eral broadly useful frameworks for stream handling exist, for example, Apache Kafka [11], Apache Flink [4], Apache Storm [19], Apache Spark Streaming [22]. Despite the fact that these frameworks are effective and non specific, they are not custom fitted towards parallel and adaptable incre-mental model preparing and induction on occasion streams

In the meantime, an expanding assortment of research addresses incre-mental (or on the web) updates of Machine Learning (ML) models: there are incremental calculations for a wide range of ML systems, for example, bolster vector machines [5], neural systems [9], or Bayesian models [21]. Plainly, a stream preparing structure supporting natural reconciliation of these calculations would be exceedingly beneficial– sparing the expenses of employing costly ML specialists to move these calculations to the stream handling frameworks.

A structural plan and programming interface for information parallel CEP that takes into consideration simple incorporation of exist-ing incremental ML calculations (cf. Area 3).

An algorithmic answer for the issues of incremental K-Means bunching and Markov display preparing with regards to peculiarity recognition in savvy manufacturing plants (cf. Area 4).

An assessment demonstrating versatility of the StreamLearner engineering and throughput of up to 500 occasions for every second utilizing our calculations for incremental ML display refreshes (cf. Segment 5).

2. CHALLENGES AND GOALS

Machine Learning calculations prepare a model utilizing a given arrangement of preparing information, e.g., building bunches, and after that apply the prepared model to take care of issues, e.g., ordering obscure occasions. Over the span of spilling information getting to be noticeably accessible from sensors, models should be powerfully adjusted. That implies, that new information is considered in the scholarly model, while old information "grows dim" and leaves the model as it ends up noticeably insignificant. This can be displayed by a sliding window over the approaching occasion streams: Events inside the window are significant for the model preparing, though occasions that drop out of the window end up plainly superfluous and ought not be re ected in the model any more. Machine Learning on sliding windows is otherwise called non-stationary Machine Learning, i.e., the issue of keeping a model refreshed as the fundamental gushing information gen-eration "process" underlies a changing likelihood appropriation. To adjust the ML demonstrate on the web, there are di erent potential outcomes. For in-position, incremental calculations change the model in a well ordered manner. The test in doing as such is to help incremental professional cessing – i.e., gushing learning. The model ought not be re-worked sans preparation for each new window, but instead incrementally be refreshed with new information while old information is expelled

Another test in ML in spilling information is that information from di erent streams may prompt autonomous models. For example, information caught in one generation machine won't not be reasonable to prepare the model of another creation machine. The test is to figure out which autonomous models should be fabricated in light of which information from which approaching occasion streams. Further, the inquiry is the manner by which to course the comparing occasions to the suitable model. At the point when these inquiries are explained, the identi ed machine learning models can be worked in parallel – empowering versatile, low-inertness, and high-throughput stream preparing.

3 STREAM LEARNER

In this area, we rst give an outline about the StreamLearner engineering, trailed by a portrayal of the simple to-utilize API for incremental machine learning and circumstance induction models.

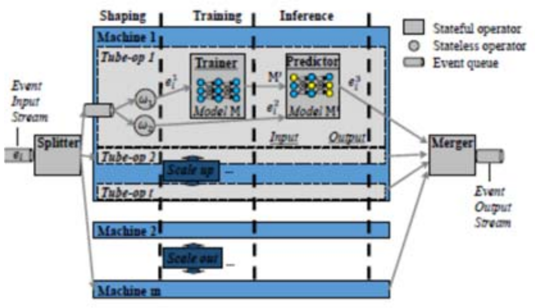


Figure 1: System Architecture

3.1 System Overview

The design of StreamLearner is given in Figure 1. Keeping in mind the end goal to paralleliz

e ML-based calculation, we have broadened the part procedure combine engineering of conventional occasion based frameworks [16–18]. The splitter gets occasions by means of the occasion input stream and advances them to free preparing units, signified as tube-operations, as per its part rationale. Each tube-operation molecularly performs ML-based incremental stream preparing by perusing an occasion from the in-line, preparing the occasion, and sending the yield occasion to the merger. The merger chooses about the nal occasions on the occasion yield stream (e.g. sorts the occasions from the di erent tube-operations by timestamp to give a reliable requesting of the occasion yield stream). Because of the free handling of occasions, the design underpins both, scale-up operations by bringing forth more strings per machine and scale-out operations by including more machines.

Each tube-operation forms an occasion in three stages: molding, prepare ing, and derivation. In the molding stage, it performs stateless pre-handling operations ω1 and ω2 (meant as shaper) to change the info occasion into proper arrangements. In the preparation stage, the stateful mentor module incrementally refreshes the model parame-ters of model M (e.g. a neural system in Figure 1) as per the client speci ed display refresh work. In the deduction stage, the refreshed model and the preprocessed occasion fill in as a contribution for the stateful indicator playing out a client de ned derivation operation and changing the refreshed model and the info occasion to a yield occasion with the model-driven forecast.

Note that the StreamLearner API does not limit application software engineers to perform preparing and surmising on di erent occasion information. Henceforth, application software engineers are allowed to utilize either disjoint subsets, or crossing subsets of occasions in the stream for preparing and derivation. In spite of the fact that it is normal practice in ML to isolate information that is utilized for preparing and derivation, despite everything we give this ex-ibility, as in genuine streams we may utilize a few occasions for both, fusing changing examples into the ML demonstrate and starting a deduction occasion utilizing the indicator. In any case, the application software engineer can likewise isolate preparing and surmising information by de n-ing the administrators in the tube-operation appropriately (e.g. creating a fake occasion as contribution for the indicator to show that no induction step ought to be performed). Besides, the application program-mer can likewise determine whether the preparation ought to occur before induction or the other way around.

3.2 Programming Model

The application developer speci es the accompanying capacities keeping in mind the end goal to utilize the StreamLearner structure in a disseminated environ-ment

3.2.1 Spli er. Given an occasion ei , the application developer de nes a stateful part work split¹ei º that profits a tuple.¹mid; tid; ei º de ning the tube-operation tid on machine mid that gets occasion ei .

3.2.2 Shaping. The stateless shaper operations ω1¹ei º and ω2¹ei º return modi ed occasions ei1 and ei2 that fill in as contribution for the coach and the indicator module. The default shaper plays out the character operation.

3.2.3 Trainer. The stateful coach operation mentor ¹ei1º restores a reference to the refreshed model question M 0. The application star grammer can utilize any kind of machine learning model as long as the model can be utilized for deduction by the indicator. On the off chance that the model M stays unaltered in the wake of handling occasion ei1, the coach must restore a reference to the unaltered model M with a specific end goal to trigger the indicator for every occasion. StreamLearner plays out a postponing technique when the application software engineer favors surmising before learning. For this situation, the tube-operation rst executes the indicator on the old model M and executes the mentor a short time later to refresh the model.

3.2.4 Predictor. The stateful indicator gets a reference to display M 0 and information (occasion) ei2 and restores the anticipated occasion

e3 = indicator M 0; e2 i¹i º

3.2.5 Merger. The stateful merger gets anticipated yield occasions from the tube-operations and returns an arrangement of occasions that is put to the occasion yield stream, i.e., merger ¹ei3º = f ¹e03; :::; ej3; :::; ei3º for j < I and any capacity f . Any aggregator work, occasion requesting plan, or ltering technique can be executed by the merger.

4 CASE STUDY: ANOMALY DETECTION IN SMART FACTORIES

In this area, we epitomize use of our Stream Learner API based on a reasonable utilize case for information investigation postured by the DEBS Grand Test 20171 [10].

4.1 Problem Description

In brilliant industrial facilities, distinguishing failing of generation machines is critical to empower programmed disappointment rectification and convenient responses to bottlenecks in the generation line. The objective of this contextual investigation is to distinguish irregularities, i.e., unusual successions of sensor occasions evaluating the condition of the generation machines. Specifically, the input occasion stream comprises of occasions transporting estimations from an arrangement of generation machines P to an irregularity identification administrator. The occasions are made by the arrangement of sensors S that screen the creation machines.We incorporate the time stamps of each deliberate sensor occasion by dening an arrangement of discrete time steps DT . Every occasion ei = ¹pi ;di ; si ; ti º comprises of a generation machine id pi 2 P that was checked, a numerical information esteem di 2 R measuring the state of the generation machine (e.g. temperature, weight, disappointment rate), a sensor with id si 2 S that has created the occasion, and a period stamp ti 2 DT putting away the occasion creation time.

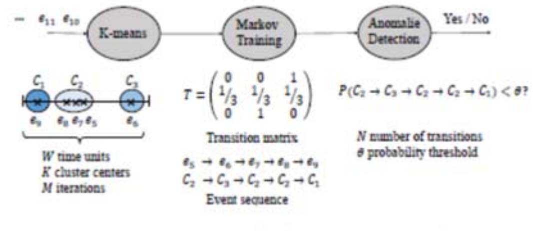


Figure 2: Case study *anomaly detection* in smart factories.

The abnormality identification administrator needs to pass three phases for each occasion producing sensor (cf. Figure 2).

To start with, it gathers all occasions ei that were created inside the last W time units (meant as occasion window) and groups the occasions ei utilizing the K-implies calculation on the numerical information esteems di for at any rate M cycles. The standard K-implies calculation iteratively relegates every occasion in the window to its nearest bunch focus (with regard to euclidean separation) and recalculates each group focus as the centroid of every single alloted occasion's numerical information esteems (in the tailing we don't dierentiate amongst occasions and their information values). In the gure, there are ve occasions e5; e6; e7; e8; e9 in the occasion window that are grouped into three bunches C1;C2;C3. With this strategy, we can describe every occasion as indicated by its state, i.e., the bunch it is appointed to.

Second, the administrator prepares a rst-arrange Markov show all together to differentiate ordinary from unusual occasion arrangements. A Markov show is a state chart, where a likelihood esteem is related to each state progress. The likelihood of a state change depends just on the present state and not on past state advances (autonomy suspicion). These probabilities are kept up in a progress grid T utilizing the accompanying strategy: (I) The Markov display comprises of K states, one state for each group. Every occasion is thought to be in the condition of the group it is doled out to. (ii) The occasions are requested concerning their opportunity stamp – from most established to most youthful. Ensuing occasions are seen as state advances. In Figure 2, the occasions can be arranged as »e5; e6; e7; e8; e9¼. The particular state advances are C2 ! C3 ! C2 ! C2 ! C1. (iii) The change lattice contains the probabilities of state advances between any two states, i.e., group focuses. The likelihood of two resulting occasions being in bunch Ci and progress into group Cj for all I; j 2 f1; :::;Kg is the relative number of these perceptions. For instance the likelihood of change from state C2 to state C1 is the quantity of occasions in state C2 that change to state C1 partitioned by the aggregate number of advances from state C2, i.e.,

P¹C1 jC2º = #C2!C1 #C2!? = 1 3.

Third, an abnormality is dened utilizing the likelihood of a grouping of watched changes with length N. Specifically, if a progression of impossible state changes is watched, i.e., the aggregate grouping likelihood is underneath the limit , an occasion is produced that demonstrates whether an abnormality has been found. The likelihood of the grouping can be figured by breaking the grouping into single state advances, i.e., in Figure 2, P¹C2 !C3 !C2 !C1º = P¹C2 !C3ºP¹C3 ! !C2ºP¹C2 !C1º. Utilizing the autonomy presumption of Markov models, we can dole out a likelihood incentive to each grouping of state progress and henceforth evaluate the probability

4.2 Formulating the Problem in the Stream Learner API

The situation is pleasantly into the StreamLearner API: for every sensor, a free ML demonstrate is liable to incremental preparing furthermore, induction steps. Along these lines, each string in the StreamLearner Programming interface is in charge of all perceptions of a solitary sensor empowering StreamLearner to screen various sensors in parallel.

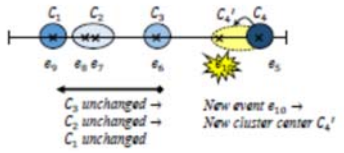


Figure 3: Saving computation time in K-Means.

4.2.1 Splitter: The splitter gets an occasion ei = ¹pi ;di ; s; ti º what's more, doles out the occasion only to the string that is in charge of sensor s (or starts formation of this mindful string in the event that it doesn't exist yet). It utilizes a straightforward hash delineate sensor ids to string ids to furnish string determination with steady time multifaceted nature amid preparing of the information occasion stream. With this strategy, we break the info stream into various autonomous sensor occasion streams (one stream for every sensor).

4.2.2 Shaper: Shapers !1 and !2 are basically character administrators that pass the occasion without changes to the separate preparing or expectation modules.

4.2.3 Trainer: The coach keeps up and refreshes the model in an incremental mold. The model is denied by means of the progress matrixT that is figured utilizing K-implies grouping and the separate state change arrangement.

Incremental K-Means.

The objective is to iteratively allot every occasion to the nearest group focus and recalculate the bunch focus as the centroid of every single allocated occasion. The standard approach is to perform M cycles of the K-implies grouping calculation for all occasions in the occasion window when activated by the landing of another occasion. Be that as it may, this technique brings about problematic runtime due to superfluous calculations that emerge in down to earth settings

A solitary new occasion in the occasion window will once in a while have a worldwide effect to the bunching. Specifically, generally assignments of occasions to bunches stay unaltered after adding another occasion to the occasion window. In this way, the animal power strategy for full reclustering can bring about colossal computational redundancies.

Performing M emphasess is superfluous, if the grouping has just united in a before emphasis M0 < M. Plainly, we ought to end the calculation as quick as could be allowed.

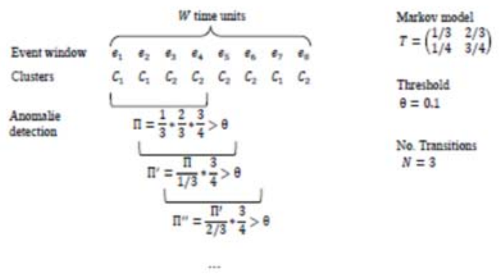
The one-dimensional K-implies issue is on a very basic level simpler than the standard NP-hard K-implies issue: an ideal arrangement can be figured in polynomial time O¹n2Kº for xed number of groups K and number of occasions in the window n [13, 20]. In this way, utilizing a universally useful K-implies calculation that backings discretionary dimensionality can bring about superfluous overhead (the exchange obetween sweeping statement, execution, and optimality).

This is represented in Figure 3. There are four bunches C1; :::;C4 furthermore, occasions e5; :::; e9 in the occasion window. A newevent e10 is arriving. Rather than recomputation of the entire bunching in every cycle, i.e., ascertaining the separation between every occasion and bunch focus, we touch just occasions that are possibly aected by a difference in the bunch focuses. For instance, occasion e10 is appointed to group C4 which prompts another group focus C04

. In any case, the following nearest occasion e6 (left side) keeps a similar group focus C3. Our essential thinking is that every occasion on the left half of the unaltered occasion e6 keeps its bunch focus as there can be no unsettling influence in the type of changed bunch focuses left-hand of e6 (just a falling group focus move is conceivable as C4 C3 C2 C1 in any stage of the calculation). A comparable argumentation can be made for the right side and furthermore for the expulsion of occasions from the window.

This thought vigorously uses the likelihood of arranging group focuses furthermore, occasions in the one-dimensional space. It diminishes normal runtime of a solitary cycle of K-implies as much of the time as it were a little subset of occasions must be gotten to. Joined with the streamlining of skipping further calculation after union in cycle M0 < M, incremental updates of the grouping can be substantially more ecient than guileless reclustering. The incremental one-dimensional bunching strategy is in a similar many-sided quality class as credulous reclustering as in the most pessimistic scenario, we need to reassign all occasions to new bunches (the arranging of occasions takes just logarithmic runtime many-sided quality in the occasion window measure per inclusion of a new occasion – henceforth the multifaceted nature is commanded by the K-implies calculation).

4.2.4 Predictor. The pointer module applies the inducing step on the changed model for each moving toward event. In this circumstance, enlistment is done through the Markov appear (i.e., the advance matrix T) to choose if an irregularity was recognized or not. We use the change network to assign a probability motivating force to a course of action of events with related states (i.e., aggregate core interests). The brute oblige procedure would process the consequence of state advance probabilities for each gathering of length N and difference it and the probability confine . Regardless, this prompts various overabundance estimations for coming about events. We display an upgraded incremental method in Figure 4. The event window contains events e1; :::; e8 masterminded by time stamps.



W time units
Event window
e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8
Clusters
C_1 C_1 C_2 C_2 C_2 C_2 C_1 C_2
Anomalie detection
$\Pi = \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{3}{4} > \theta$
$\Pi' = \frac{\Pi}{1/3} \cdot \frac{3}{4} > \theta$
$\Pi'' = \frac{\Pi'}{2/3} \cdot \frac{3}{4} > \theta$
...
Markov model
$T = \begin{pmatrix} 1/3 & 2/3 \\ 1/4 & 3/4 \end{pmatrix}$
Threshold
$\theta = 0.1$
No. Transitions
$N = 3$

Figure 4: anomaly detection.

Every occasion is alloted to a group C1 or C2 bringing about an arrangement of state changes. We utilize the change network of the Markov model to decide the likelihood of each state change. We ascertain the likelihood of the state change arrangement as the result of all state advances (the state autonomy property of Markov models). For example the likelihood of the rst three state advances is = P^{1}C1 jC1^{o} P^{1}C2 jC1^{o} P^{1}C2 jC2^{o} = 1 3 2 3 3 4 = 1 4 which is bigger than the edge = 0:1. Presently we can without much of a stretch figure the likelihood of the following state progress grouping of length N by isolating by the rst progress likelihood of the grouping (i.e., P^{1}C1 jC1^{o} = 1 3) and duplicating with the likelihood of the new state change (i.e., P^{1}C2 jC2^{o} = 3 4). Thus, the aggregate likelihood 0 of the following state change succession is 0 = 1 3 3 4 = 9 16 > . This strategy lessens the quantity of augmentations to N +2^{1}W □N^{o} as opposed to N^{1}W □N^{o}. At long last, the indicator issues an irregularity discovery occasion to the merger (Yes/No).

4.2.5 Merger. The merger sorts all oddities occasions w.r.t. time stamp to guarantee a predictable yield occasion stream utilizing the same system as in GraphCEP [17]. This strategy guarantees a monotonic increment of occasion time stamps in the yield occasion stream.

5 EVALUATIONS

In this area, we give our examinations Stream Learner on the DEBS Grand Challenge 2017 informational collection with 50,000 sensor information occasions.

Exploratory Setup:
We utilized the accompanying two processing situations. (I) A scratch pad with 43:5 GHz (8 strings, Intel Core i7-4710MQ), 8 GB RAM (L1 Cache 256 KB, L2 Cache 1024 KB, L3 Reserve 6144 KB), and 64 Bit bolster. (ii) An in-house shared memory framework with 32 2:3 GHz (Quad-Core AMD Opteron(tm) Processor 8356), and 280 GB RAM (L1d reserve 64 KB, L1i store 64 KB, L2 reserve 512 KB, L3 store 2048 KB), and 64 Bit bolster.

Adjusting the window sizeW:
In Figure 5a, we demonstrate the outright throughput of StreamLearner on the y-pivot and dierent window sizesW on the x-pivot utilizing the note pad for a dierent number of strings. Unmistakably, bigger window measure prompts bring down throughput as computational overhead develops. We standardized this information in Figure 5c to the interim »0; 100¼ to think about the relative throughput upgrades for the different number of strings.

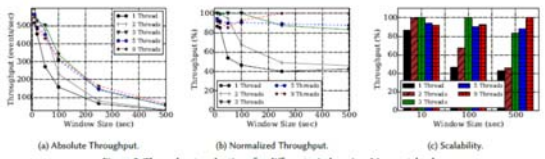


Figure 5: Throughput evaluations for different window sizes W on notebook.

Unmistakably, the bene t of multi-threading emerges just for bigger window sizes due to the consistent dissemination overhead that can not be repaid by expanded parallelism on the grounds that each string has just minimal computational assignments between purposes of synchronization (on the splitter and on the merger). General adaptability is estimated in Figure 5a. It can be seen that StreamLearner scales best for information parallel issues with generally little synchronization overhead in contrast with the computational assignment. For little window sizes (e.g.W = 10), throughput does not increment with expanding number of laborers. Be that as it may, for direct to huge window sizes, scaling the quantity of laborer strings increasingly affects the relative throughput: scaling from one to nine strings expands throughput by 2:5.

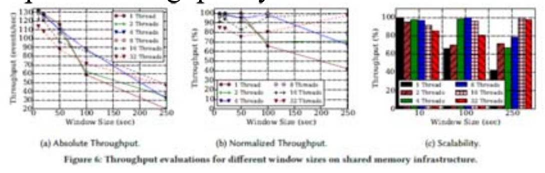


Figure 6: Throughput evaluations for different window sizes on shared memory infrastructure.

In Figure 6a, we rehashed the test on the mutual memory framework. The rst perception is that the single strung tests are four times slower contrasted with the scratch pad foundation because of the more seasoned equipment. All things considered, in Figure 6b, we can see obviously that the relative throughput diminishes when utilizing a low as opposed to a high number of strings (e.g. for bigger window sizesW > 100). In Figure 6c, we measure versatility enhancements of up to 60%. By the by, it can be additionally observed that it isn't generally ideal to utilize a high number of strings – regardless of whether the issue is very parallelizable.

Adjusting the quantity of bunches K:

In Figure 7, we plot the outright throughput for a shifting number of bunches and different strings. We xed the window measure toW = 100. Of course, an expanding number of bunches prompts lessened throughput due to the expanded computational intricacy of the bunching issue. Clearly, expanding the quantity of strings builds the throughput up to a specific point. This is predictable with the ndings above.

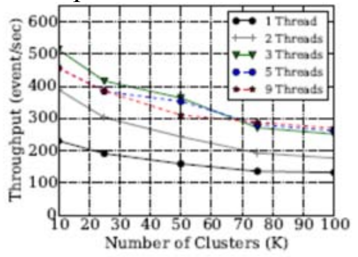


Figure 7: Throughput for varying number of clusters *K*.

6 CONCLUSION

Stream Learner is a conveyed CEP framework and API customized to adaptable occasion identification utilizing Machine Learning on gushing information. In spite of the fact that our API is universally useful, StreamLearner is particularly appropriate to information parallel issues – with numerous occasion sources causing different examples in the occasion streams. For these situations, StreamLearner can improve standard CEP frameworks with capable Machine Learning usefulness while scaling incredibly well due to the pipelined incremental preparing and deduction ventures on autonomous models.

7 REFERENCES

[1] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2009. The planar k-means

problem is NP-hard. In International Workshop on Algorithms and Computation. Springer, 274–285.

[2] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for largescale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 135–146.

[3] Christian Mayer, Muhammad Adnan Tariq, Chen Li, and Kurt Rothermel. 2016. GrapH: Heterogeneity-Aware Graph Computation with Adaptive Partitioning. In Proc. of IEEE ICDCS.

[4] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. 2015. Predictable Low- Latency Event Detection with Parallel Complex Event Processing. Internet of Things Journal, IEEE 2, 4 (Aug 2015), 274–286.

[5] Ruben Mayer, Christian Mayer, Muhammad Adnan Tariq, and Kurt Rothermel. 2016. GraphCEP: Real-time Data Analytics Using Parallel Complex Event and Graph Processing. In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS ’16). ACM, New York, NY, USA, 309–316. DOI:https://doi.org/10.1145/2933267.2933509

[6] Ruben Mayer, Muhammad Adnan Tariq, and Kurt Rothermel. 2017. Minimizing Communication Overhead in Window-Based Parallel Complex Event Processing. In Proceedings of the 11th ACM International Conference on Distributed and Eventbased Systems (DEBS ’17). ACM, New York, NY, USA, 12. DOI:https://doi.org/10. 1145/3093742.3093914.

[7] Shen Furao, Tomotaka Ogura, and Osamu Hasegawa. 2007. An enhanced selforganizing incremental neural network for online unsupervised learning. Neural Networks 20, 8 (2007), 893–903.

[8] Vincenzo Gulisano, Zbigniew Jerzak, Roman Katerinenko, Martin Strohbach, and Holger Ziekow. 2017. The DEBS 2017 grand challenge. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS ’17, Barcelona, Spain, June 19 - 23, 2017.

[9] Jay Kreps, Neha Narkhede, Jun Rao, and others. 2011. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB. 1–7.

[10] Narayanan C Krishnan and Diane J Cook. 2014. Activity recognition on streaming sensor data. Pervasive and mobile computing 10 (2014), 138–154.