# Online pattern discovery in distributed, high-dimensional, streaming data under the YOLO principle.

Andreas Grammenos[*]
University of Cambridge
ag926@cl.cam.ac.uk

Cecilia Mascolo[†]
University of Cambridge
cm542@cl.cam.ac.uk

Jon Crowcroft [†]
University of Cambridge
jac22@cl.cam.ac.uk

## INTRODUCTION

Over the past few decades the amount of data being produced and transferred over internet has been ever increasing at an almost exponential rate but so has the need to analyze them. This increase, makes storing the data in its entirety for OLAP processing intractable both for practical reasons (e.g. storage costs) as well as due to the inherent computational complexity of processing said data. Also, even if some data are kept, most of the times it is infeasible to provide exact answers within a reasonable amount of time, especially when working in the online setting where we only get to look at the incoming data only *once*, which is known as the YOLO[1] principle. Additionally, we argue that in many use-cases it is acceptable to provide approximate answers that are close the exact ones but can be computed significantly faster while also requiring orders of magnitude less storage. In addition, to topic of finding and tracking emerging, hidden, or periodic patterns can be directly applicable many use-cases example of which are low latency stock market monitoring, temperature trend detection, DDoS attack detection and generally any interesting unbounded information signal.

Of course, this is not a new concept and there have been numerous works either looking at the related theoretical or systems aspects but almost never both; we argue for such algorithmic tools in order to be practical and be actually used in practice both aspects need to be considered at the same time. In particular this work aims to be, in its final form, a combination of i) algorithmic tools and ii) systems architecture that will work in tandem to provide answers to the iii) top-$k$ pattern tracking problem in the streaming, online setting at a massive scale.

## OBJECTIVES

Concretely, our framework will try to satisfy the following criteria:

*O1: Ease-of-use.* Our implementation will use a widely known streaming framework in order to maximize its reach and will hide its complexity by using higher level primitives provided.

*O2: Performance.* : Aim to provide answers fast, usually with minimal latency while also being resilient to missing values.

*O3: Flexibility.* Will be able to adjust the tracked patterns either at will, or based on a given function $f$ which can be used to determine the number of patterns to keep in each node.

*O4: Scalability.* Will be able to handle a vast amount of data, and be able to distribute the load automatically over the available nodes thus achieving a linear horizontal scaling.

[*]PhD Student, Submitter
[†]PhD Supervisor
[1]You Only Look Once

## PROBLEM STATEMENT

Given $m$ groups of data-streams that each one consists of $\{n_1, ..., n_m\}$ co-evolving numeric data-streams, we want to devise a solution that will solve the following two problems:

- Incrementally find the top-$k$ most dominant patterns $L_i \in \mathbb{R}^{1\times k}$ within a single group $i \in [1, m]$ using $F_L$ function.
- Efficiently combine the each one of the $m$ local patterns $L_i$ to find the representative global patterns $G \in \mathbb{R}^{1\times k}$ using $F_G$ function of all monitored streams in each group.

### Preliminaries

In this problem we model each data-stream using the *time-series* model and as such a stream data-source, for our purposes is an unbounded sequence of tuples in time-order. For specificity we assume that such data-streams produce one tuple per time-unit. To find such patterns more efficiently a sound technique is to find a method to distribute the workload amongst many workers. This is performed by splitting the input data-streams into *groups* and incrementally finding the patterns in each of the groups. Finding the patterns in each of the groups is considerably less expensive than finding all the patterns at once; this step is called *local-pattern-discovery*. Then by having discovered each of the groups' *local-patterns* find efficiently the *global-patterns* that are representative for all of our monitored streams.

Additionally, we view the original data-streams as points in a high-dimensional space[2], where as stated above, produce one tuple per time-unit. After grouping the streams; each of the groups' local-patterns are extracted using low-dimensional projections of the original points.

### Problem Formalization

Before we introduce these two aforementioned functions we have to first describe the data-stream distribution amongst each group. Let a group be denoted as $S$ then each $i$-th group $S_i$ is comprised out of an unbounded sequence of $n_i$-dimensional vectors where $n_i$ is the number of data-streams contained in the group $S_i$ with $1 \leq i \leq m$ and $m$ being the total number of monitored groups. $S_i$ can also be represented as a matrix with $n_i$ columns and an unbounded number of rows. The *intersection* of matrix $S_i$, defined as $S_i(t, l)$ is the $t$-th row and the $l$-th column of $S_i$; this represents the value of the $l$-th stream recorded at time $t$ in the $i$-th group. Finally using the definitions described above we are now able to define the functions for monitoring per-group *local-patterns* $F_L$ and the aggregated *global-patterns* $F_G$.

[2]The dimension can be user adjustable, from 1 to thousands

Function $F_L$ definition for *local-pattern* calculation:

$$F_L : (S_i(t+1,:), B) \longrightarrow L_i(t+1,:) \tag{1}$$

$F_L$ takes as input $S_i(t+1,:) \in \mathbb{R}^{1 \times n_i}$ vector which contains current values for each data-stream monitored by the group $i$ at time $(t+1)$ and the block-size that we current have, usually this is a static value throughout the execution of our system.

The function $F_L$ has to incrementally maintain the *local-patterns* $L_i \in \mathbb{R}^{1 \times d}$ which is then propagated to the aggregators in order to produce the *global-patterns*. In a similar fashion like $F_L$ the definition for the *global-pattern* detection function $F_G$ follows in full.

$$F_G : (L_1(t+1,:), L_m(t+1,:)) \longrightarrow G(t+1,:) \tag{2}$$

The $F_G$ function takes as arguments all the *local-patterns* $L_i, i \in [1, m]$ for each one of the $m$ groups that were generated at time $(t+1)$ and produces a *global-pattern* vector $G(t+1,:)$ at each time-step which is then propagated to each one of the $m$ groups and is also our final output that holds the top-$k$ most dominant trends of all monitored streams.

An interesting *relaxation* of the problem, as it leads to potential novel research avenues, is to propagate values *only* at certain points e.g. when something *"interesting"* happened; this could expressed as a particular event occurrence, certain value changes, or could be given by an arbitrary, node specific, monitor function $f_m$.

## PATTERN DISCOVERY USING ONLINE PCA

Principal component analysis is a linear transformation, specifically it uses an orthogonal transformation to convert a set of observed possibly correlated values into a set of values of linearly uncorrelated variables that are called *principal components*. The number of principal components of a *lossless* transformation is the same as the number of the original variables before performing PCA. The dimensionality reduction comes when we are allowed to perform a *lossy* PCA transformation, this can be done easily as the principal components can be sorted based on their significance using the eigenvalues; this allows to "drop" a number of non-significant principal components in order to perform the dimensionality reduction. Hence, PCA is a great way to reduce the dimensionality of data and would ideally suit our needs; yet the classic method for computing PCA is limited by the prohibitive cost of forming the covariance matrix, which typically is when having a stream of $n$-dimensional vectors a $n \times n$ dense matrix that in turn requires $n^2$ space. The quadratic cost of space is, as one might imagine, barred for large datasets or in *streaming* scenarios and poses a major bottleneck for its potential applications.

The output of PCA when operating on streaming vectors of $n$-dimensions is a set of $k \in [1, n]$, $n$-dimensional principal components which span the subspace created by PCA transformation. Naturally a *lossless* application of PCA means that $k = n$ and would result in a $n \times n$ matrix but a relaxation of the *lossless* decomposition is expected and we would be allowed to have a significantly reduced number of principal components. Mitliagkas et al. in [5] proposed a novel algorithm with strict theoretical bounds on approximation quality that required $O(kn)$ space which by definition is the lowest possible when performing PCA. They base their method in the classic Power-Method as described in [1] and adapt into a block-wise stochastic variant with great results. The intuition behind their algorithm stemmed from the fact that most stochastic methods for PCA approximation had a variable and possible large variance at each step and hence the standard concentration of inequalities would give vacuous bounds. Their proposed method used a block-wise algorithm that had a variance reduction step built in; in essence they would update the basis $Q_\tau$ of PCA once every block using a lossy QR-decomposition while within each block they would average-out the noise thus reducing the variance contained in processed values. For this particular setting each node incrementally calculates its own $Q_r$ basis approximation which is kept locally and at any time requires $O(kn)$ space.

Finally, we are now ready to define our notion of a pattern, which we define as the *projection* the input to the current basis approximation $Q_r$. This operation produces a vector in $\mathbb{R}^k$, where $k$ is the number of patterns tracked, ordered in terms of significance.

## PRELIMINARY IMPLEMENTATION DESIGN

As per (*O1*) we elected to use for the purposes of our implementation Apache Flink [2], that although is a relatively new streaming framework is widely used, mature, and has good support within the community. In addition, it provides a highly-scalable and flexible infrastructure that we leverage to achieve (*O2*). The algorithmic framework that we developed was based initially on the *streaming* PCA methods described in [5, 6] and was briefly outlined above. In order to satisfy (*O3)* performed the following two integral changes in order to satisfy this goal, i) created a user-adjustable *thresholding* method for which we would increase/decrease the tracked patterns during execution and ii) instead of having to transfer the whole subspace matrix (size $O(kn)$) we only transfer its projections to the current input (size $O(k)$) – which is our local pattern vector in $\mathbb{R}^k$ as described previously. Moreover, this streaming computation of the projections can happen in multiple remote nodes independently and due to the fact that PCA is a linear transformation the summaries can be trivially merged in aggregation nodes; this property enables us to efficiently parallelise our workload (*O4*). Finally, since the actual data are not stored which has the side-benefit of inherently preventing the leakage of global information in case a system node is compromised.

## EARLY RESULTS

The modified method shown in [5, 6] with our aforementioned contributions is able to beat, in terms of accuracy, and pattern discovery existing frameworks [7] while showing to be more scalable and resilient to faults or missing values. Additionally, our method is computationally efficient as with our initial implementation, with no significant language level optimizations, we are able to concurrently process, in the online setting, more than 5k streams on a single thread when using a modern processor[3]. In all of our experiments and in order to have a direct comparison we used the motes dataset which was provided from [3] and the baseline performance calculated using [7].

[3]Intel Xeon X5690

## ONGOING WORK

We are in the process of porting and extending our single-node proof-of-concept into the distributed environment of Apache Flink and evaluating its performance at scale. Our goal is to meet and exceed our set objectives which hopefully, will enable the discovery of patterns from a vast amount of streaming data that was not possible before.

## FUTURE DIRECTIONS

Future directions would be to finish the implementation of our system, assess its performance along with solving any potential engineering and/or algorithmic challenges along the way. Finally, it would be worth improving current missing value handling mechanism with ideas stemming from [4] and experimenting the *relaxed* setting of the problem as defined previously, where we only propagate the patterns iff an "interesting" event occurred.

## REFERENCES

[1] Raman Arora, Andrew Cotter, Karen Livescu, and Nathan Srebro. 2012. Stochastic optimization for PCA and PLS. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on.* IEEE, 861–868.

[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[3] Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30.* VLDB Endowment, 588–599.

[4] Armin Eftekhari, Laura Balzano, Michael B Wakin, and Dehui Yang. [n. d.]. SNIPE for Memory-Limited PCA From Incomplete Data: From Failure to Success. ([n. d.]).

[5] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory limited, streaming PCA. In *Advances in Neural Information Processing Systems.* 2886–2894.

[6] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2014. Streaming pca with many missing entries. *Preprint* (2014).

[7] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st international conference on Very large data bases.* VLDB Endowment, 697–708.