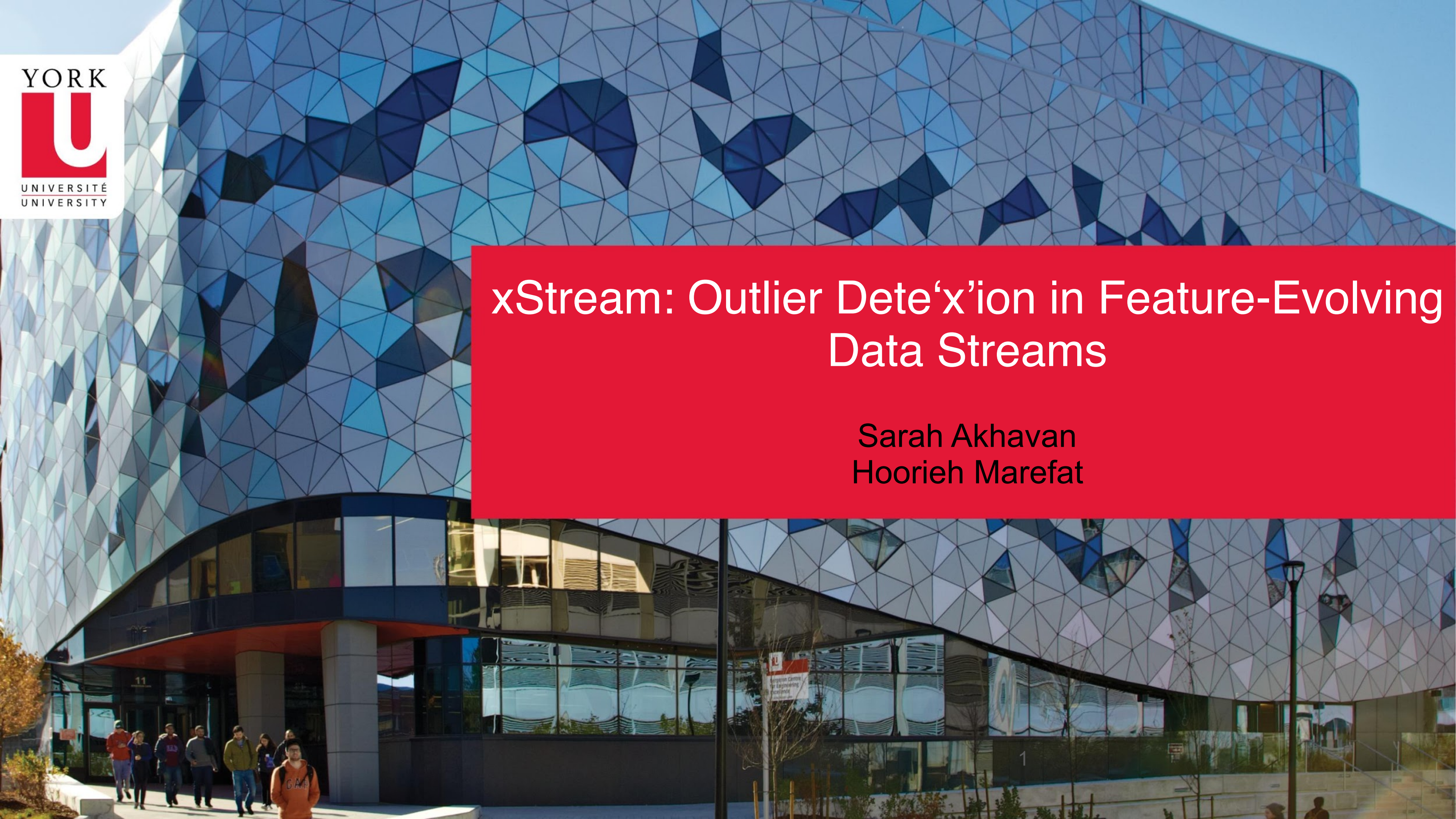


xStream: Outlier Dete‘x’ion in Feature-Evolving Data Streams

Sarah Akhavan
Hoorieh Marefat

Paper Description

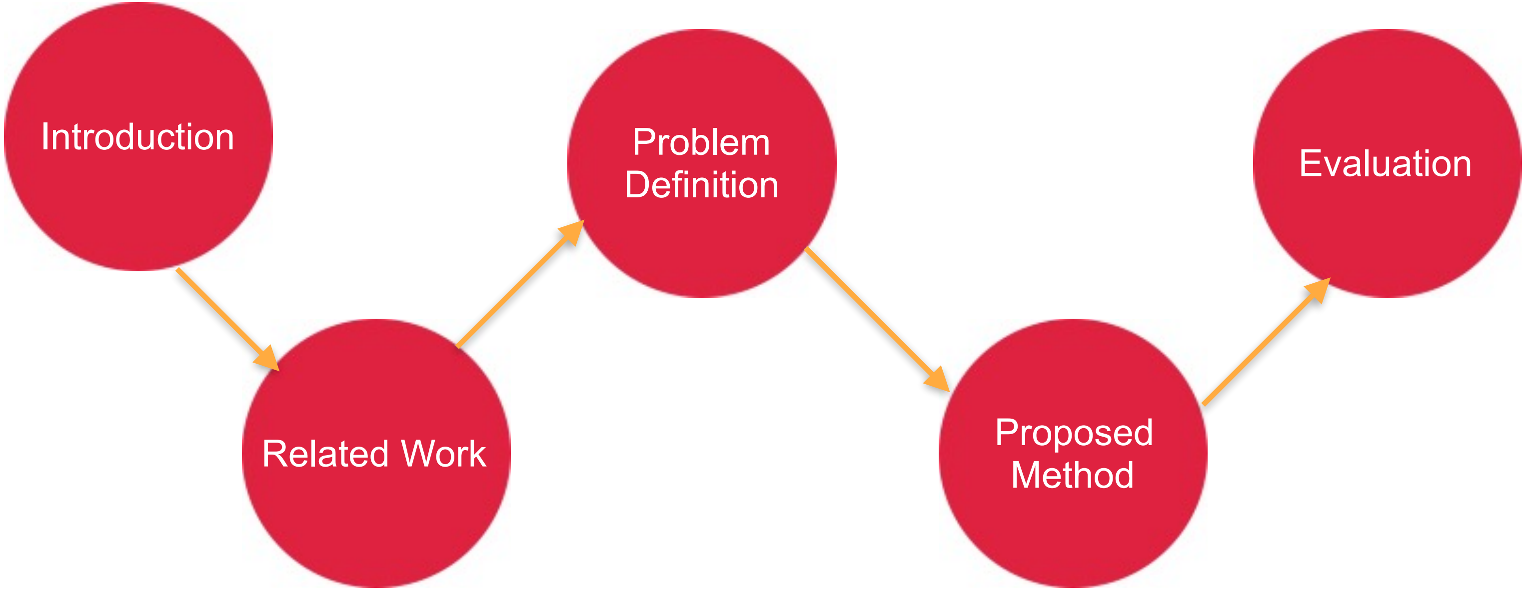
xSTREAM: Outlier Dete'x'ion in Feature-Evolving Data STREAMS

Emaad Manzoor
H. John Heinz III College
Carnegie Mellon University
emaad@cmu.edu

Hemank Lamba
School of Computer Science
Carnegie Mellon University
hlamba@cs.cmu.edu

Leman Akoglu
H. John Heinz III College
Carnegie Mellon University
lakoglu@andrew.cmu.edu

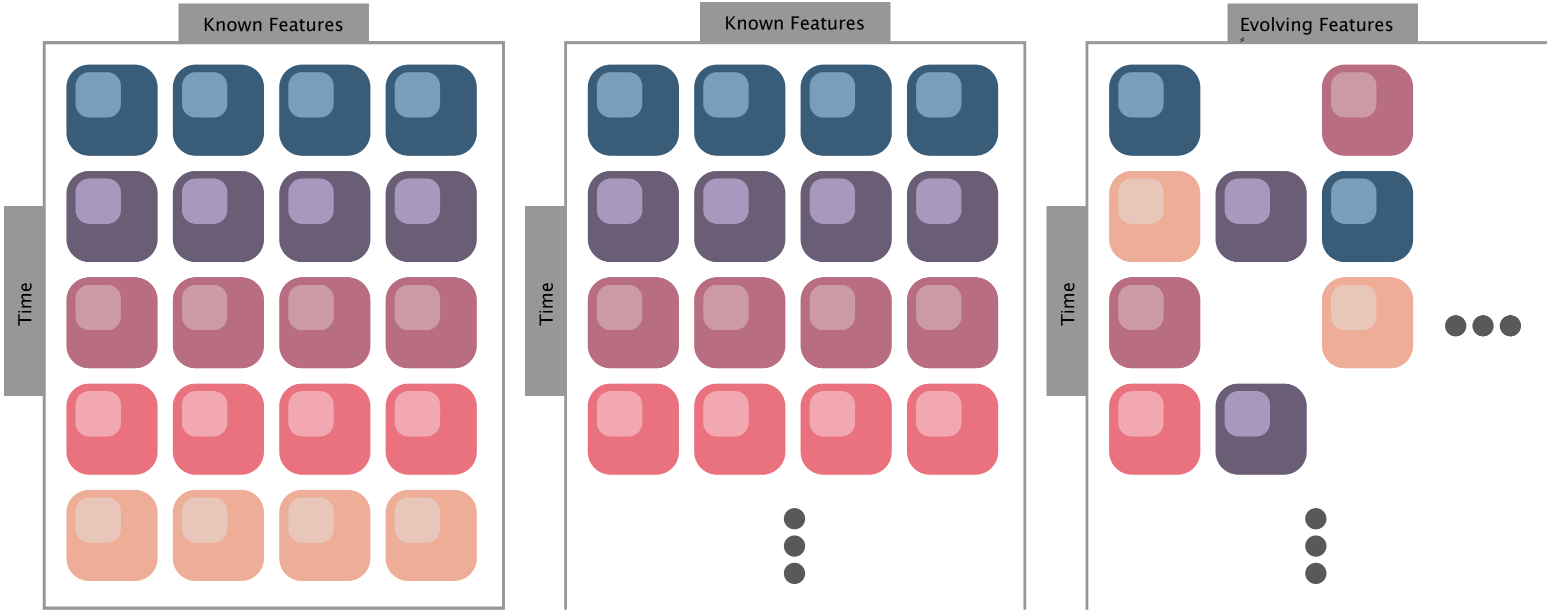
Agenda


Introduction
Related Work
Problem Definition
Proposed Method
Evaluation

Data Types

- static
- row streaming
- feature-evolving stream

Data Types



static data row-streaming data feature-evolving stream

Key Challenges

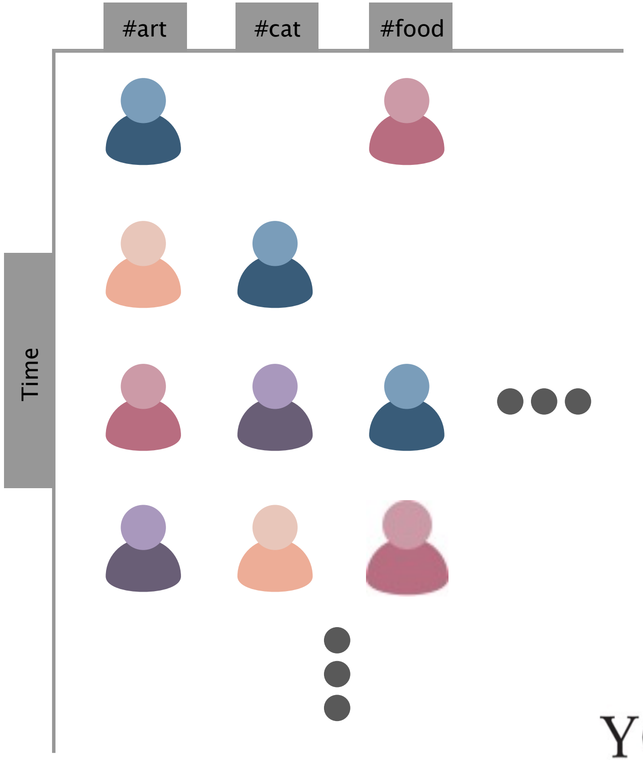
- we cannot simply process and discard points
    - feature values may change
    - new features may emerge
- dynamically allocated space for growing n and d
- fastness of the technique

Other Challenges

- non-stationarity of the data stream
- curse of dimensionality
- outliers at multiple scales or different subspaces

Practical Cases For This Setting

- User Monitoring (e.g: Twitter)
- Data Center Monitoring
- Customer Behavior Tracking



xStream

- **constant memory** approach
- processing each element in **constant time**
- measures outlierness in **multiple scales**
- handles **non-stationary**
- accommodates **static data** and **row-streaming data**

Related Work

- Ensemble Methods
  - feature subspace selection

- Streaming Methods
  - ensemble methods that partition the representation space
  - RS-Hash

Comparing xStream with state-of-the-art outlier detection techniques

Methods/ Properties	LOF [10]	Feat.Bag. [22]	LOCI [25]	HiCS [21]	iForest [23]	HS-Stream [31]	STORM [6]	LODA [26]	RS-Hash [29]	RS-Forest [32]	xSTREAM
Static	✔	✔	✔	✔	✔	✔		✔	✔	✔	✔
Streaming						✔	✔	✔	✔	✔	✔
Multi-scale			✔								✔
Subspaces		✔		✔	✔	✔		✔	✔	✔	✔
Projections								✔			✔
Evolving feature space											✔
Evolving points/ feature values											✔

Notation

- We have an incoming stream of elements

$$\mathcal{D} = \{e_t\}_{t=1,2,\ldots}$$

- With each $e_t = (id, f, \delta)_t$ showing an update to a point
  - *id*: identifies each data point
  - *f*: is a feature name (a string)
  - δ: magnitude of the update

Problem Definition

- **Problem**: *Outlier Detection in Feature-Evolving Data Streams*
- **Given** a stream $\mathcal{D} = \{e_t\}_{t=1,2,\ldots}$ of triples $e_t = (id, f, \delta)_t$
- **Compute and maintain** an outlier score for each evolving point such that outliers are scored higher than non-outlier points at any time *t*.

Overview of the Steps

- Projecting high-dimensional feature space to a low-dimensional one
- Estimate density of the neighbouring area of each data point
- Giving outlierness score to each data point

Proposed Method: xStream

- The method is built on the following components:
  1. StreamHash: subspace-selection and dimensionality reduction via sparse random projections
  2. Half-Space Chains: an efficient ensemble to estimate density at multiple scales
  3. Extensions to handle non-stationarity and evolving data points in the stream

Random Projection

- Random projections are an efficient and effective method of reducing data dimensionality
- In high-dimensional data, outliers often lie in low-dimensional subspaces => looking for outliers in selected subspaces of the data
- Here a variant of random projections is used
  - Sparse random vectors with only 1/3 of the vector components being non-zero.

StreamHash

$$\boldsymbol{y}[i] = \sum_{f_j \in \mathcal{F}} h_i(f_j)\, \boldsymbol{x}[j], \quad i = 1, \ldots, K.$$

$$\boldsymbol{y}_{id}[i] = \boldsymbol{y}_{id}[i] + h_i(f)\, \delta, \quad i = 1, \ldots, K.$$

$$h_i[f] = \sqrt{\frac{3}{K}} \begin{cases} -1 & \text{if } a_i(f) \in [0, 1/6) \\ 0 & \text{if } a_i(f) \in [1/6, 5/6) \\ +1 & \text{if } a_i(f) \in [5/6, 1] \end{cases}$$

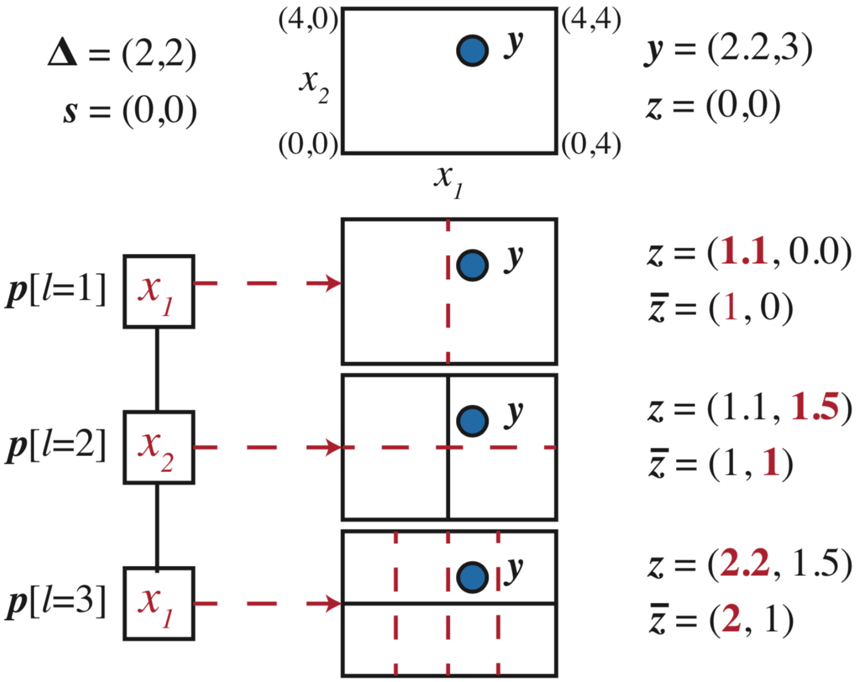
Approximating Density

- To detect density-based outliers, approximate the density of each point by counting the number of its neighbours lying within some radius r.
- Two issues with performing neighbourhood-counting directly:
  - Sensitive to the choice of scale ⇨ Compute neighbours at multiple scales Via half-space chains
  - The number of neighbours at any scale tends to zero as the dimensionality increases ⇨ Dimensionality reduction via HashStream

Half-Space Chain

$$C = \{\boldsymbol{p}, \boldsymbol{\Delta}, \boldsymbol{s}, \mathcal{H}\}$$

$$C = \{C_1 \ldots, C_M\}$$



How To Compute Z?

$$\boldsymbol{z}[p] \quad = \quad \frac{\boldsymbol{y}[p] + \boldsymbol{s}[p]/2^{o(p,l)-1}}{\boldsymbol{\Delta}[p]/2^{o(p,l)-1}}, \quad \forall\, p \in \mathcal{P}$$

$$\bar{\boldsymbol{z}} \quad = \quad \lfloor \boldsymbol{z} \rfloor$$

Multi-Scale Outlier Scoring

$$S(\boldsymbol{y}) = \frac{1}{M} \sum_{C \in C} s_C(\boldsymbol{y})$$

Issues of Stream Progress

- The distribution of points may change as the stream progresses, causing bin-counts constructed in the past to no longer represent the current distribution of the data.
- Additionally, triples in the data stream may update previously seen points.

Handling Non-Stationarity

- Non-stationarity is handled by maintaining separate bin-counts for an alternating pair of windows containing ψ points each, termed as current and reference windows.
- On the arrival of $(\psi + 1)^{th}$ new point, reference counts are replaced with current counts, and current counts are set to zero to begin processing the next window.

Handling Evolving Data Points

- Points may evolve by receiving updates in the stream to either an existing feature or to a new, previously unseen feature.
- In either case, it is required that the existing projected point y resides in main memory so as to update it quickly without accessing the disk.
  - A fixed-size cache of N projected points is maintained in memory.
  - Least-Recently-Updated (LRU) eviction protocol is used for the cache

Time and Space Complexity

- Time complexity: $O(KmDM)$
- Space complexity: $O(MmLD + NK)$
    - xStream maintains
        - M half-space chains and
        - N evolving (projected) points

Evaluation

- Static Setting
- Row-stream
- Evolving Stream

Datasets Used For Evaluation

Name	Evolving $\mathcal{F}$?	Evolving points?	n or $\|\mathcal{D}\|$	d	No. of outliers
gisette	No	No	3850	4970	351
isolet	No	No	4886	617	389
letter	No	No	4586	617	389
madelon	No	No	1430	500	130
cancer	No	No	385	30	28
ionosphere	No	No	242	33	17
telescope	No	No	13283	10	951
indians	No	No	538	8	38
Spam-SMS	Yes	No	5574	8442	747
Spam-URL	Yes	No	2.4M	3.2M	792K
Attk-Flash	Yes	Yes	63.1M	1.1M	2.8M
Attk-Java	Yes	Yes	89.7M	1.1M	29.5M

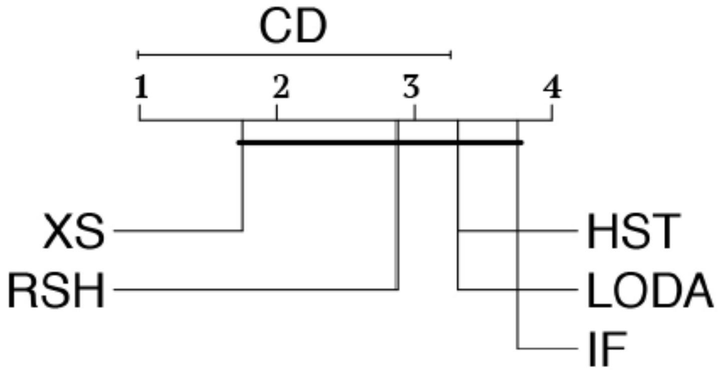
Static Stream

- iForest
- HS-Tree
- LODA
- RS-Hash

Friedman Test

Dataset	*iForest*	*HS-Trees*	*RS-Hash*	*LODA*	xStream
cancer	0.617 ± 0.021	0.646 ± 0.033	0.619 ± 0.030	0.826 ± 0.013	0.845 ± 0.008
ionosphere	0.705 ± 0.006	0.706 ± 0.007	0.764 ± 0.032	0.642 ± 0.067	0.848 ± 0.018
telescope	0.367 ± 0.008	0.392 ± 0.012	0.391 ± 0.012	0.322 ± 0.007	0.344 ± 0.009
indians	0.142 ± 0.003	0.146 ± 0.002	0.156 ± 0.007	0.177 ± 0.008	0.216 ± 0.010
gisette	0.078 ± 0.002	0.080 ± 0.002	0.084 ± 0.007	0.087 ± 0.003	0.090 ± 0.003
isolet	0.099 ± 0.003	0.097 ± 0.005	0.108 ± 0.004	0.089 ± 0.004	0.112 ± 0.006
letter	0.093 ± 0.001	0.092 ± 0.002	0.104 ± 0.004	0.094 ± 0.006	0.122 ± 0.005
madelon	0.110 ± 0.003	0.101 ± 0.013	0.092 ± 0.005	0.101 ± 0.010	0.097 ± 0.004
Avg Rank	3.75	3.3125	2.875	3.3125	1.75

Nemenyi Test



Row-Stream

- HS-Stream
- LODA
- RS-Hash

Window size ψ	*HS-Stream* MAP	*HS-Stream* OAP	*LODA* MAP	*LODA* OAP	*RS-Hash* MAP	*RS-Hash* OAP	xSTREAM MAP	xSTREAM OAP	xSTREAM-1K MAP	xSTREAM-1K OAP
1%	0.480 ± 0.178	0.416	0.090 ± 0.028	0.076	0.291 ± 0.129	0.171	0.505 ± 0.138	0.422	0.522 ± 0.153	0.430
5%	0.492 ± 0.179	0.416	0.082 ± 0.014	0.077	0.216 ± 0.034	0.195	0.455 ± 0.135	0.406	0.493 ± 0.134	0.415
10%	0.430 ± 0.024	0.419	0.081 ± 0.010	0.080	0.174 ± 0.017	0.164	0.444 ± 0.037	0.433	0.448 ± 0.037	0.436
25%	0.363 ± 0.024	0.359	0.080 ± 0.001	0.080	0.203 ± 0.014	0.201	0.409 ± 0.009	0.404	0.435 ± 0.013	0.429

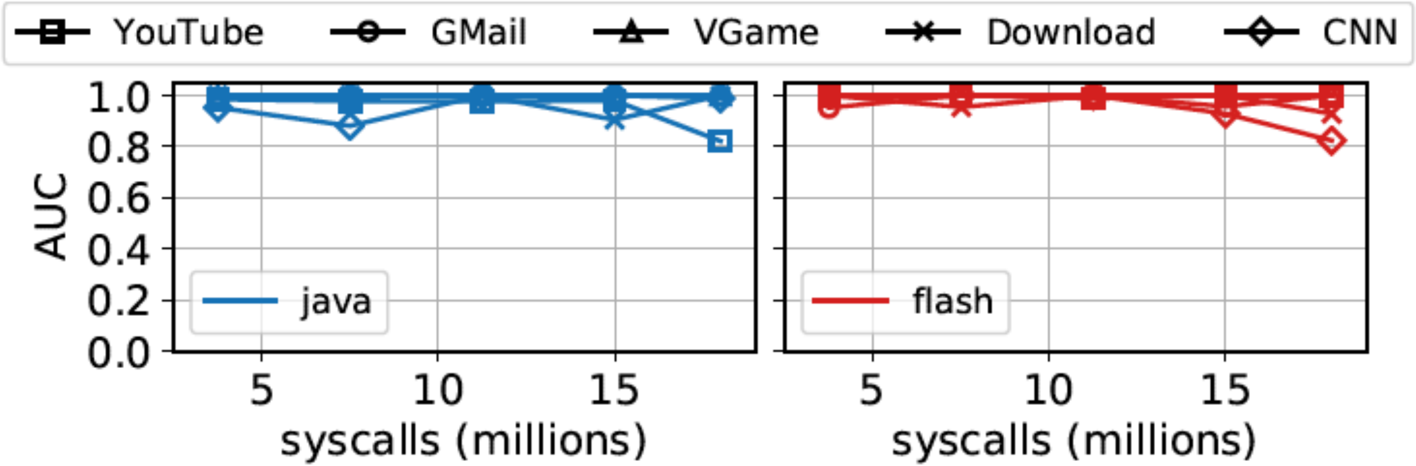
Mean average precision (MAP) and overall average precision (OAP) on Spam-SMS.

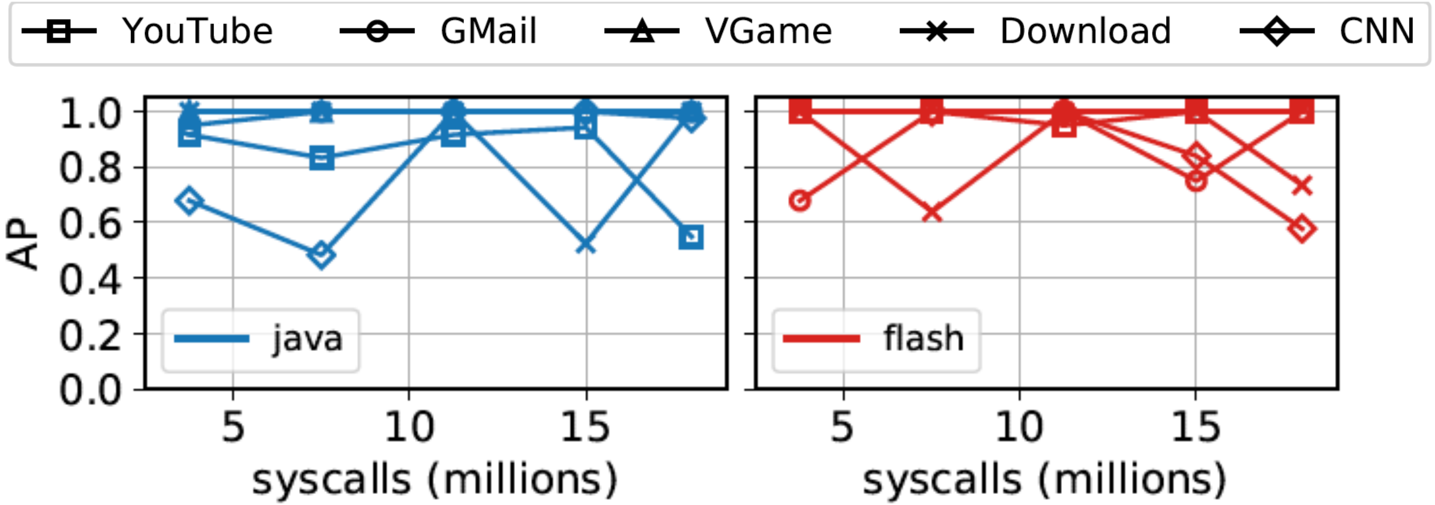
Window size ψ	*HS-Stream* MAP	*HS-Stream* OAP	*LODA* MAP	*LODA* OAP	*RS-Hash* MAP	*RS-Hash* OAP	xSTREAM MAP	xSTREAM OAP	xSTREAM-1K MAP	xSTREAM-1K OAP
1 day	0.331 ± 0.055	0.330	0.329 ± 0.059	0.331	0.358 ± 0.061	0.357	0.436 ± 0.083	0.437	0.451 ± 0.106	0.452
3 days	0.339 ± 0.058	0.329	0.307 ± 0.055	0.328	0.357 ± 0.046	0.356	0.478 ± 0.078	0.479	0.508 ± 0.064	0.509
5 days	0.336 ± 0.059	0.329	0.321 ± 0.044	0.328	0.357 ± 0.038	0.356	0.472 ± 0.050	0.472	0.493 ± 0.055	0.496
7 days[1]	—	—	0.303 ± 0.040	0.321	0.355 ± 0.036	0.356	0.497 ± 0.053	0.502	0.533 ± 0.049	0.530

[1] *HS-Stream* exceeds the available memory on a 1 TB machine.

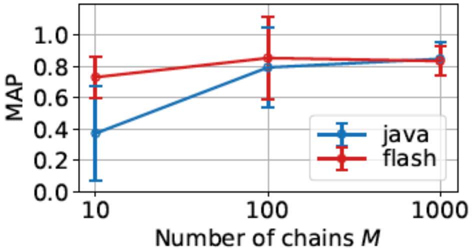
Mean average precision (MAP) and overall average precision (OAP) on Spam-URL.

Evolving Stream

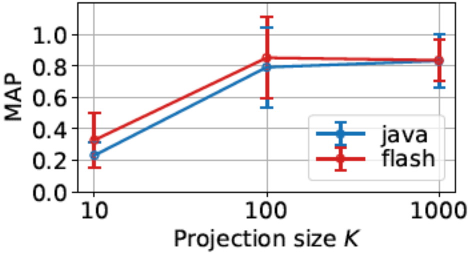




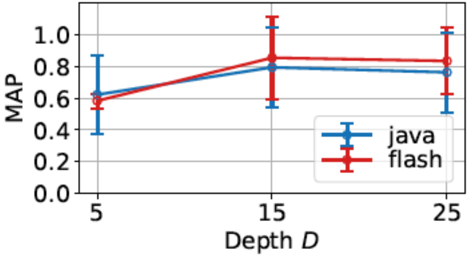
YouTube
GMail
VGame
Download
CNN
AP
1.0
0.8
0.6
0.4
0.2
0.0
java
flash
5
10
15
syscalls (millions)



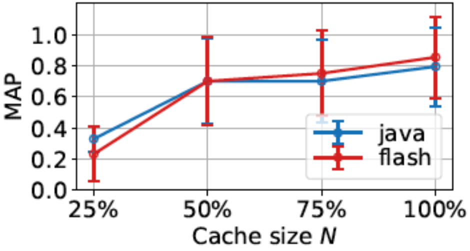
(a) MAP vs. ***M***



(b) MAP vs. ***K***



(c) MAP vs. ***D***



(d) MAP vs. ***N***