XStream

1 Histogram Forests (HF)

Fitting. For each histogram $i = 1, \ldots, T$:

1. **Initialize the random projection matrix A_i.** This may be (i) sparse with 2/3 zeros, or (ii) Gaussian random projections (which demonstrate better performance, but are computationally expensive).
2. **Project the data.** $Y = XA_i^T$.
3. **Sample a shift.** $b_i \sim \text{Uniform}(0, w)$.
4. **Bin the data.** For each projected data point $Y_j = (y_{j1}, \ldots, y_{jK})$, its bin vector is given by $\bar{Y}_j = (\lfloor \frac{y_{j1}+b_i}{w} \rfloor, \ldots, \lfloor \frac{y_{jK}+b_i}{w} \rfloor)$.
5. **Store bin counts.** In an exact hashtable or count-min sketch H_i, increment the count $H_i(\bar{Y}_j)$ for $j = 1, \ldots, N$.

Scoring. The anomaly score of a point X_j is the negative of the average of $H_i(\bar{Y}_j)$ across all histograms i.

Histogram Forest Parameters	
K	Dimension of projected data
T	Number of histograms
w	Bin width
Other variables	
X	$N \times D$ data matrix
Y	$N \times K$ projected data matrix
A_i	$K \times D$ projection matrix
b_i	Shift $\sim$ Uniform$(0, w)$
H_i	Hashtable/count-min sketch

Table 1: Notation.

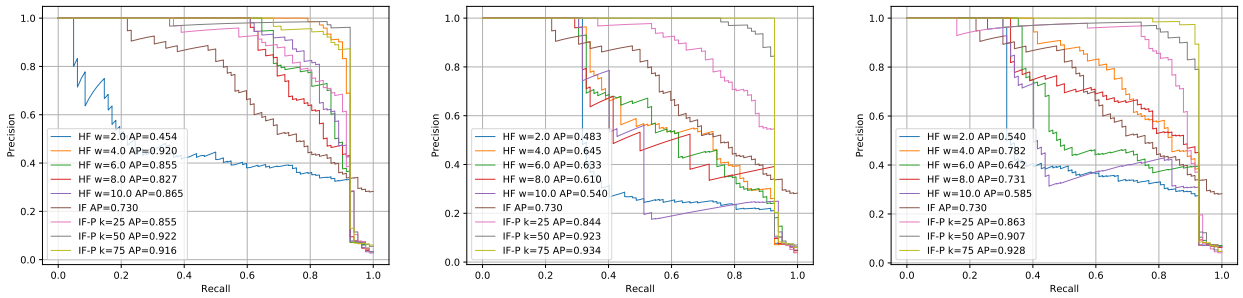
1.1 HF Results

Methods compared in Figure 1 (on the synthetic data with noise):

- **HF:** Histogram forest with varying bin width w, number of projected dimensions K and number of histograms T.
- **IF:** iForest with 100 trees.
- **IF-P:** iForest with 100 trees and data projected to k dimensions using sparse random projections, with varying k.

Observations from Figure 1:

1. HF with $K = 25, T = 100, w = 4.0$ is on par with iForest with 100 trees on the data projected to $k = 50$ or $k = 75$ dimensions, and better than iForest for $k = 25$.
2. HF with $K = 50, T = 100$ has poor performance for all w. This was our configuration setting for the previous histogram experiments (with some minor variations): we considered them a failure without trying a *lower* value of K. This suggests that K as a measure of data approximation quality may not be the right interpretation.
3. HF with $K = 50, T = 200$ improves performance over $T = 100$ slightly, for all w.



(a) $K = 25, T = 100$ (b) $K = 50, T = 100$ (c) $K = 50, T = 200$

Figure 1: Precision-recall curves on synthetic data with 100 noisy dimensions.

1.2 Interpretation — Approximate Nearest Neighbors

The binned data point in our histogram is identical to a point "hashed" using 2-stable distributions (Datar et al. 2004, 3.1), which extends LSH to solve the approximate nearest neighbors problem in L_2. In this method, each point x is "hashed" using the function $h(x) = \lfloor \frac{x^T a + b}{w} \rfloor$ where a is a random Gaussian vector, b is a random shift and w is the bin width. The probability that two points x and y hash to the same value is proportional to $\|x - y\|_2$.

Each point is hashed K times, and points that agree on all K values are considered candidate nearest neighbors. However, as K grows, it becomes less likely that even nearby points agree on all K values; hence, the process is repeated T times, and the points agreeing on all K values at in least one of the T trials are considered candidate nearest neighbors.

Each configuration of K, T and w results in a certain *probability gap* for points to be considered nearest neighbors. Specifically, this probability gap is defined by the 4-tuple (R, p_1, cR, p_2):

- The probability of two points with distance $\leq R$ being hashed to the same values is $\geq p_1$.
- The probability of two points with distance $\geq cR$ being hashed to the same values is $\leq p_2$.

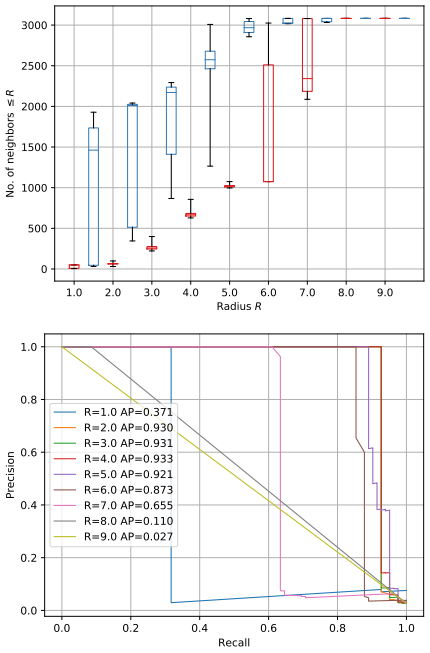


Figure 2: (Top) Number of neighbors within radius R, for anomalous (red) and benign (blue) points. (Bottom) PR curves using exact nearest neighbor counts.

Hence, our anomaly score for a point can be viewed as the negative of the approximate number of points within a sphere of some radius R centered at that point. The optimal R for anomaly detection on a given dataset is data-dependent (Fig. 2). Hence, the optimal K, T, w are also data-dependent. This explains why a $K = 25$ histogram forest may perform better than a $K = 50$ one.

1.3 Data-dependent Parameter Tuning

The anomaly detection performance is sensitive to picking the right parameters K, T and w (equivalent to picking a radius R and the LSH approximation probabilities p_1 and p_2). There have been some approaches to auto-tuning the LSH parameters based on the data. However, they were primarily designed for nearest-neighbor search and do not work for data streams. Could we design a data structure for data streams that is tailored for anomaly detection?

- LSH-Forest (Bawa et al. 2005). The earliest approach; this is a very simple approache that builds a trie using each bit of the K-element sketch. Demonstrated for the Jaccard and cosine distances (since their LSH values are binary). A theoretical analysis was recently published (Andoni et al. 2017).
- Parameter-free LSH (Ahle et al. 2017). The most recent approach, designed for Euclidean distances. This builds a "multi-level LSH" data structure that tries multiple values of the projection size K, and finds the best level for each nearest-neighbor query.

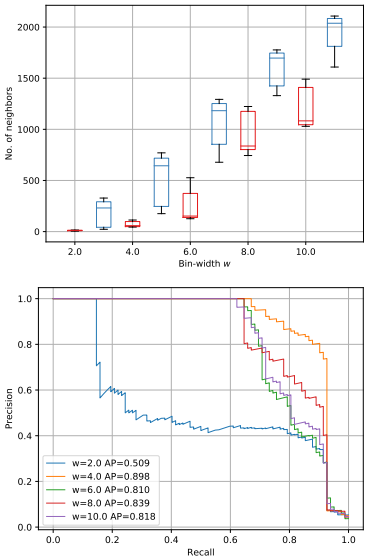


Figure 3: (Top) Distribution of approximate nearest neighbor counts via HF for different w (Bottom) PR curve for HF. $K = 25, T = 100$.

1.4 Relationship to Chains

2 Chains

Pre-initialization. This is done once, and is unchanged for all chains.

1. **Initialize the random projection matrix** A**.**
2. **Project the data.** $Y = XA^T$.
3. **Set** $\Delta_f =$ half the range of projected feature f.
4. **Sample a shift** b_{if} for each chain i and feature f.

Chain fitting. For each chain $i = 1, \ldots, C$:

For each depth $d = 1, \ldots, D$:

1. **Sample a dimension** $f_d \in \{1, \ldots K\}$ and store the dimension sampled at each depth. Let $0 \leq c(f, d) \leq d$ be the number of times dimension f has been sampled in the chain until depth d.
2. **Bin the data.** Let $Y_j = (y_{j1}, \ldots, y_{jK})$ be a projected data point. Denote by the $Z_{jd}[f]$ the "unfloored bin index" of feature f for point j at depth d:

$$Z_{jd}[f] = \frac{y_{jf} + b_{if}/2^{c(f,d)-1}}{\Delta_f/2^{c(f,d)-1}} \quad \forall f = 1, \ldots K \tag{1}$$

At depth d, the bin indices $\bar{Y}_{jd}$ are logically defined as follows, for $f = 1, \ldots, K$:

$$\bar{Y}_{jd}[f] = \begin{cases} 0, & \text{if } c(f,d) = 0 \\ \lfloor Z_{jd}[f] \rfloor & \text{if } c(f,d) > 0 \end{cases} \tag{2}$$

In practice, the bins are computed recursively at each depth as follows:

$$\bar{Y}_{jd}[f] = \begin{cases} 0 & \text{if } c(f,d) = 0 \\ \lfloor Z_{jd}[f] \rfloor & \text{if } c(f,d) = 1 \\ \lfloor 2 \times Z_{jd-1}[f] - \frac{b_{if}}{\Delta_f} \rfloor & \text{if } c(f,d) > 1 \end{cases} \tag{3}$$

The recurrence is derived as follows:

$$x_k = \frac{y + b/2^k}{\Delta/2^k} = \frac{y + b/2^{k-1}}{\Delta/2^k} - \frac{b/2^k}{\Delta/2^k} = 2\left(\frac{y + b/2^{k-1}}{\Delta/2^{k-1}}\right) - \frac{b}{\Delta} = 2x_{k-1} - \frac{b}{\Delta} \tag{4}$$

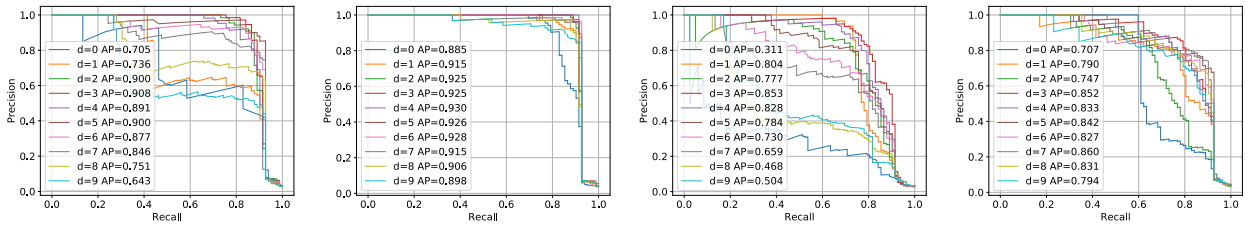
3. **Store bin counts.** In an exact hashtable or count-min sketch H_{id}, increment the count $H_{id}(\bar{Y}_j)$ for $j = 1, \ldots, N$.

Scoring at depth d**.** The anomaly score of a point X_j at depth d is $-\text{average}_i(H_{id}(\bar{Y}_j))$.

Chain Parameters	
K	Dimension of projected data
C	Number of chains
D	Chain depth
Δ_f	Initial/maximum bin width
Other variables	
f	A feature index in $1, \ldots, K$
X	$N \times D$ data matrix
Y	$N \times K$ projected data matrix
A	$K \times D$ projection matrix
b_{if}	Shift $\sim$ Uniform$(0, \Delta_f)$
H_{id}	Hashtable/count-min sketch

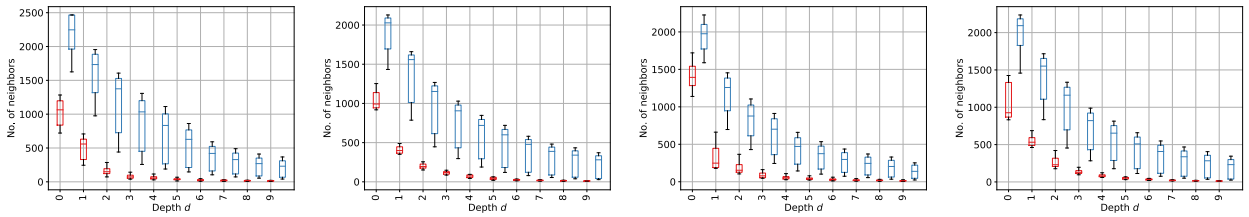
Table 2: Notation.

2.1 Results with bin-counts as anomaly scores



(a) $K = 50, C = 10$ (b) $K = 50, C = 50$ (c) $K = 25, C = 10$ (d) $K = 25, C = 50$

Figure 4: Precision-recall curves on synthetic data with 100 noisy dimensions, $D = 10$.



(a) $K = 50, C = 10$ (b) $K = 50, C = 50$ (c) $K = 25, C = 10$ (d) $K = 25, C = 50$

Figure 5: No. of neighbors in the same bin for anomalous (red) and benign (blue) points.

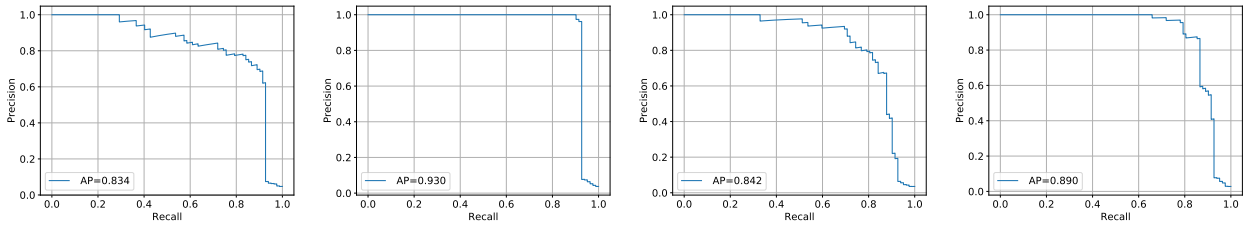
2.2 Results with LOCI scores as anomaly scores

Previously, we scored each point X_j using chain i at depth d using the number of neighbors lying in the same bin, $H_{id}(\bar{Y}_j)$. We modify this to obtain a LOCI score $L_{id}(\bar{Y}_j)$ as:

$$\bar{H}_{id}(\bar{Y}_j) = H_{id}(\bar{Y}_j) \times 2^d \tag{5}$$

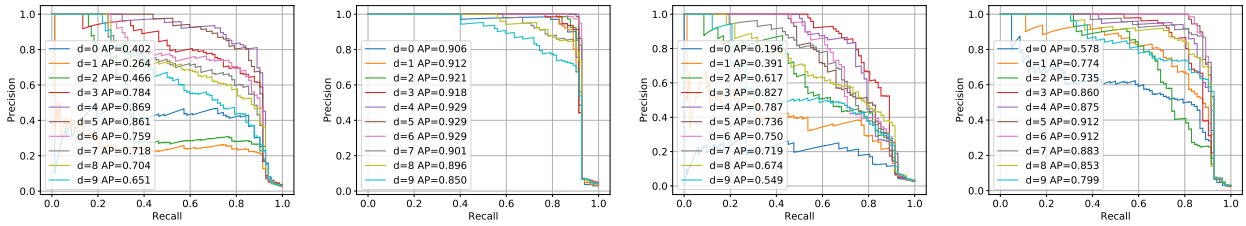
$$L_{id}(\bar{Y}_j) = \bar{H}_{id}(\bar{Y}_j) - \frac{1}{N-1} \sum_{k \neq j} \bar{H}_{id}(\bar{Y}_k) \tag{6}$$

Then the anomaly score of a point X_j is $-\text{average}_i \left(\min_d L_{id}(\bar{Y}_j) \right)$.



(a) $K = 50, C = 10$ (b) $K = 50, C = 50$ (c) $K = 25, C = 10$ (d) $K = 25, C = 50$

Figure 6: Precision-recall curves on synthetic data with 100 noisy dimensions, $D = 10$.



(a) $K = 50, C = 10$ (b) $K = 50, C = 50$ (c) $K = 25, C = 10$ (d) $K = 25, C = 50$

Figure 7: Precision-recall curves with LOCI scores at different depths d.

2.3 Stability across runs

Chain Parameters	Mean AP	Standard Deviation
$K = 50, C = 10$	0.824	0.064
$K = 50, C = 50$	0.930	0.001
$K = 25, C = 10$	0.742	0.123
$K = 25, C = 50$	0.901	0.019

Table 3: Chain AP stability across 5 runs, $D = 10$.

2.4 Number of unique bin IDs with depth

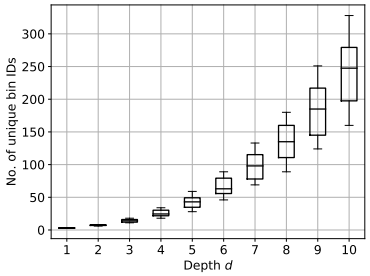


Figure 8: No. of unique bins for $C = 100$ chains, $K = 50, d = 10$.

2.5 Scores for different anomaly types

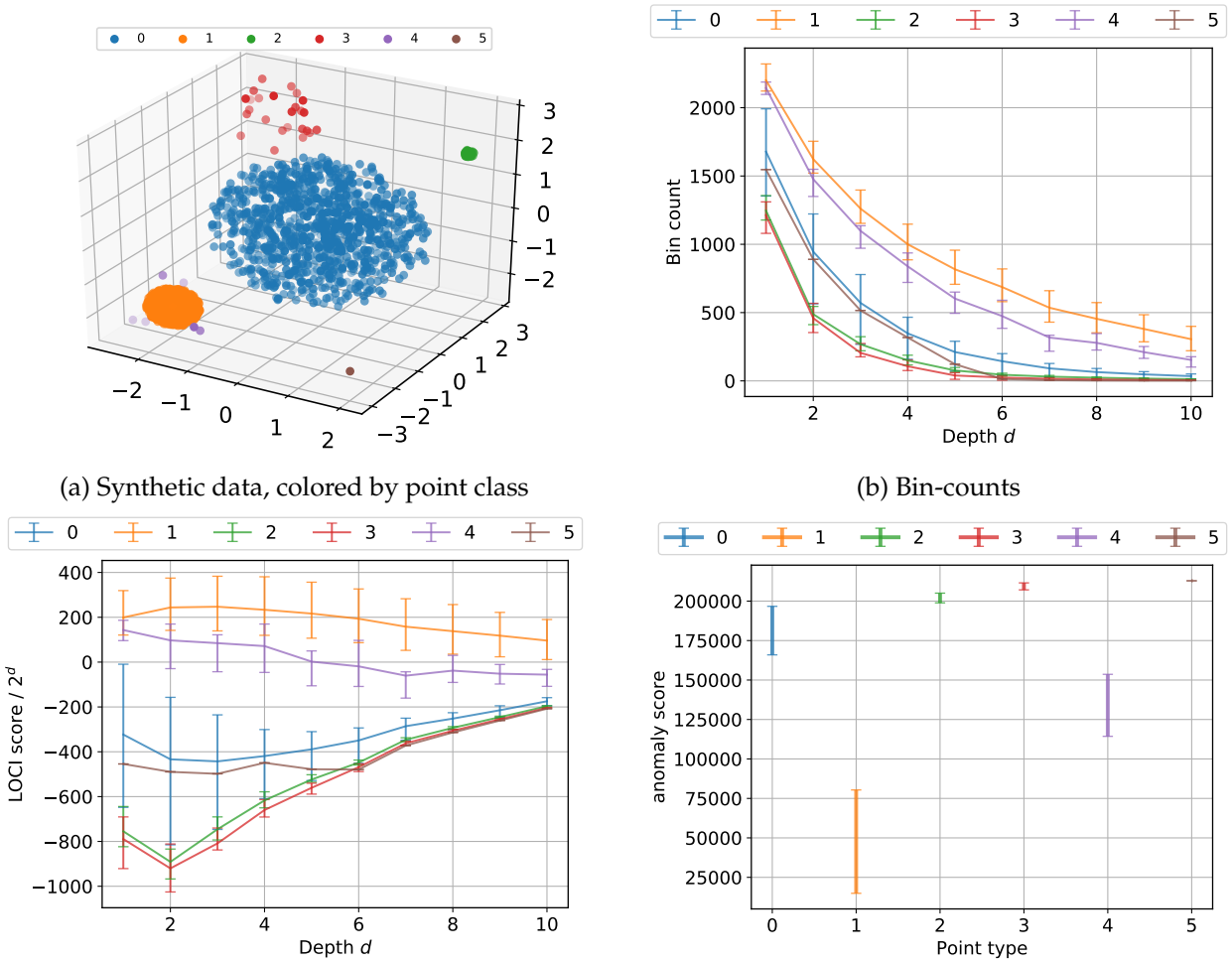
Fig. 9 illustrates how the anomaly scores vary for different classes of points, shown in Fig. 9(a):

Class	No. of points	Description	Color
Class 0	1000	Benign sparse points	Blue
Class 1	2000	Benign dense points	Orange
Class 2	50	Anomalous dense points	Green
Class 3	25	Anomalous sparse points	Red
Class 4	6	Locally anomalous points	Purple
Class 5	1	Single anomalous point	Brown

From Fig. 9(b) conforms to intuition: as the depth in the chain increases, the average number of points lying in the same bin (across chains) decreases for all classes of points. This is due to two factors: (i) increasing the number of bin dimensions, and (ii) decreasing the bin width. Point classes lying in denser regions have higher bin counts.

Fig. 9(c) shows the LOCI scores (eq. 6) at each depth d, scaled up by 2^d for visual clarity. The LOCI score of a point is simply its bin count minus the mean bin count of all other points at that depth. Hence, as d increases and the bin counts of all points tend to 1, the LOCI score tends to 1.

Fig. 9(d) shows the anomaly scores. The order of anomalousness across point classes is inversely proportional to their global density.



(a) Synthetic data, colored by point class

(b) Bin-counts

(c) LOCI scores (mean-adjusted bin-counts)

(d) Anomaly scores (negative of the minimum LOCI score across all depths)

Figure 9: Score distributions for different point classes (see §2.3 for details). Error bars indicate the median, 5th and 95th percentile scores across all points. Chains with $K = 50, C = 100, d = 10$.

2.6 Streaming random projections

Let the data dimensionality be D and projection dimensionality be K. Given a density s, the very sparse random projections method (Li et al. 2006) generates K D-dimensional vectors containing elements r_{km} as follows ($k = 1, \ldots, K, m = 1, \ldots, D$):

$$r_{km} = \begin{cases} -\frac{1}{\sqrt{Ks}} & \text{with probability } \frac{s}{2} \\ 0 & \text{with probability } 1 - s \\ \frac{1}{\sqrt{Ks}} & \text{with probability } \frac{s}{2} \end{cases}$$

Instead of storing these vectors, we store K hash functions. Each hash function h_k hashes a string w_m (the feature name) to a hash value $h_k(w_m)$, which serves as a plug-in replacement for r_{km} without explicitly storing the random vectors.

The implementation of h_k ensures that the hash values follow the probability distribution prescribed above.

The function $h_k(w_m)$ is implemented as follows:

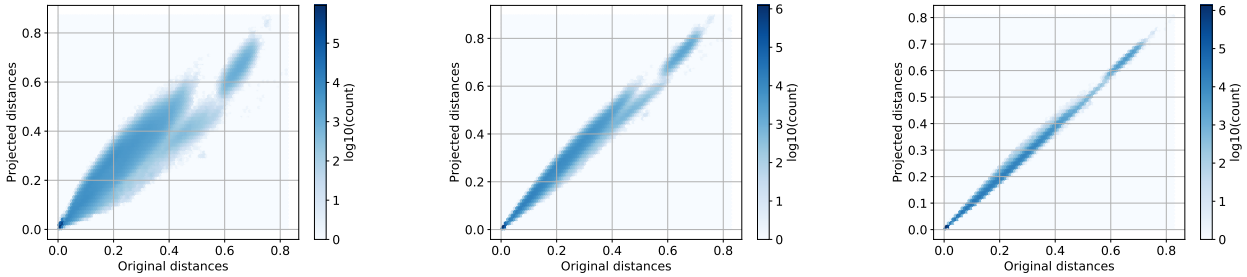
1. $b = g_k(w_m)$, where g_k is drawn from a universal family and b is a 32-bit integer. Having more bits in b provides a more precise approximation to the desired probability distribution, but we found 32 to be sufficient.
2. $n = b/(2^{32} - 1)$ is a number between 0 and 1. With a good universal family to generate g_k, n follows distributions $P_{g_k}(n) \sim \text{Uniform}(0, 1)$ and $P_{w_m}(n) \sim \text{Uniform}(0, 1)$.
3. n is used to return a number in $\{\frac{1}{\sqrt{Ks}}, 0, \frac{1}{\sqrt{Ks}}\}$ with the desired probability distribution:
   (a) If $0 \leq n < s/2$, return $-\frac{1}{\sqrt{Ks}}$.
   (b) If $s/2 \leq n < s$, return $\frac{1}{\sqrt{Ks}}$.
   (c) If $s \leq n \leq 1$, return 0.

It is very important that g_k be drawn from a universal (or almost-universal) string hash family with good empirical randomization properties. SipHash (Aumasson et al. 2012) and MurmurHash3[1] were found to perform well, whereas multiplicative hashing does not.

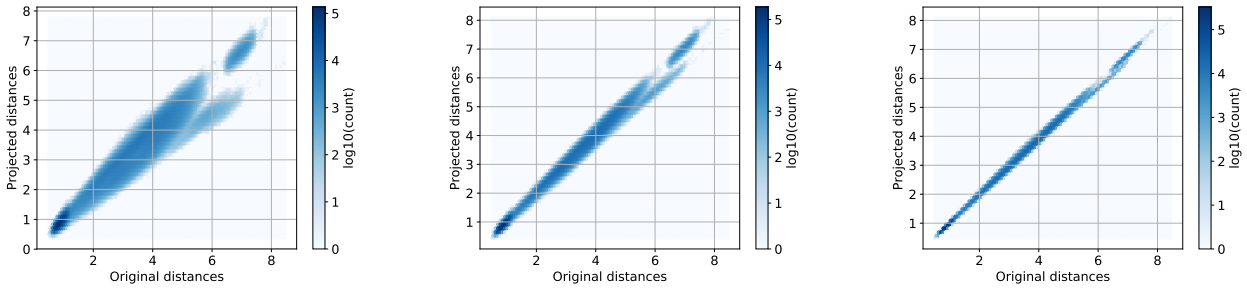
Fig. 10 illustrates the empirical distortion of this streaming random projection method on the synthetic data, for both the Euclidean and cosine distances. Fig. 11 illustrates the distortion using the standard, static random projections.

When used as a plug-in replacement for sparse random projections in XStream on the synthetic data, there was no significant drop in average precision (mean and standard deviation reported for 5 runs): AP = 0.823 ± 0.088 for $K = 50, C = 50$; AP = 0.916 ± 0.009 for $K = 100, C = 50$; AP = 0.927 ± 0.003 for $K = 100, C = 100$.

[1] `https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp`

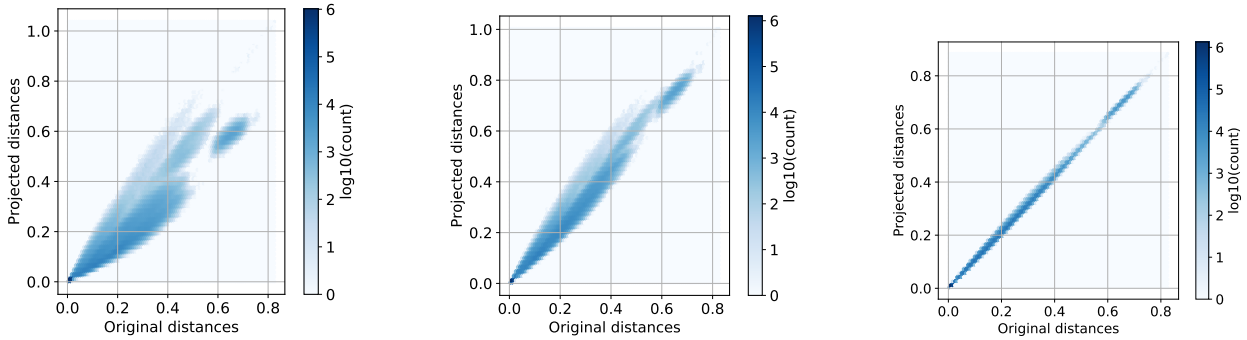


(a) $K = 50$, cosine. (b) $K = 100$, cosine. (c) $K = 1000$, cosine.

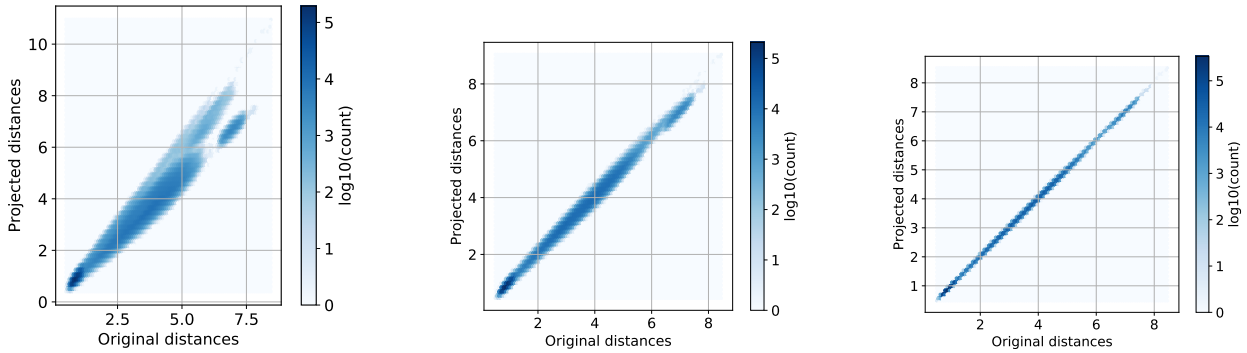


(d) $K = 50$, Euclidean. (e) $K = 100$, Euclidean. (f) $K = 1000$, Euclidean.

Figure 10: Distortion of streaming random projections for the Euclidean and cosine distance.



(a) $K = 50$, cosine. (b) $K = 100$, cosine. (c) $K = 1000$, cosine.



(d) $K = 50$, Euclidean. (e) $K = 100$, Euclidean. (f) $K = 1000$, Euclidean.

Figure 11: Distortion for static random projections for the Euclidean and cosine distance.

2.7 Dynamic chains with windows

3 Static evaluation

Dataset	Number of samples	Dimensionality
High-dimensional Datasets		
gisette	7000	4970
isolet	7797	616
letter	7797	616
madelon	2600	500
Low/medium-dimensional Datasets		
cancer	568	30
ionosphere	350	33
telescope	19020	10
indians	768	8

Table 4: Datasets used for the static evaluation.

3.1 Datasets

The datasets used are listed in Table 4 to generate the subsequent datasets used in the static evaluation. They were chosen based on (TODO: selection logic).

3.2 Generating anomalies for static evaluation

We used 2 different procedures to generate the datasets on which to benchmark, one for the low/medium dimensional datasets and one for the high-dimensional datasets. They are described below:

1. *Low/medium-dimensional datasets:* We append new columns containing Gaussian noise to the original dataset, thereby increasing its dimensionality with noisy features. The number of noisy features is quantified as a percentage of the original dataset dimensionality. If the original dataset mean and standard deviation are μ and σ, the noisy columns contained Gaussian noise with mean $\rho\mu$ and standard deviation $\rho\sigma$, where ρ is the noise factor. We consider appending 100%, 1000%, 2000% and 5000% noisy features, and use $\rho \in \{0.01, 0.1, 0.2, 0.25\}$.

2. *High-dimensional datasets:* We first discard all the original anomalous points. Now for the remaining nominal points, we first choose 10% of the features at random and mark them as important features. We further proceed with choosing 5% of the samples at random as important samples. Now, in this sub-block, which contains only important samples and important features, we add Gaussian noise to the original data. Again, Gaussian noise is used, parameterized by the mean, and the standard deviation of the entire nominal dataset. Further this noise is scaled according to the user-specified signal to noise ratio, and added to the original sub-block of our dataset and features.

3.3 Competing methods and hyperparameter settings

XStream (XS) was used with $K = 100$ projections, $C = 100$ chains and depth $d = 15$. We implemented and compared against the following competing methods and parameter settings:

- **iForest (IF)**: We use it with recommended sample size of 256 or the entire dataset, whichever is minimum. The `hlim` is set to 15 to be comparable to HS-Trees, and number of components is set to 100.
- **HS-Trees (HST)**: Use it with the recommended setting of 15 max-depth and 100 trees.
- **LODA**: We set the sparsity factor to $\frac{1}{\sqrt{(d)}}$. We allow LODA to select best width for histograms, but fix the number of histograms to 100.
- **RS-Hash (RSH)**: We use 1000 sampling points and 100 components.

3.4 Baseline Results

Table 12 lists the average precision of XStream and the competing methods on the generated datasets.

A Friedman test for differences in the best-performing method across all datasets rejected the null hypothesis that the difference ranking between methods is not statistically significant with $p = 2.2 \times 10^{-16} < 0.05$.

Hence, we perform a Wilcoxon signed-rank test to compare methods pairwise and obtain a ranking over all datasets. Table 5 contains p-values from the Wilcoxon signed-rank test. Each cell (i, j) in the table contains the p-value of the hypothesis that method i has a higher AP than method j is statistically significant. Hence, **XS** > **LODA** > **RSH** > **IF** > **HST** is statistically significant at $p = 0.05$.

	IF	**RSH**	**LODA**	**HST**	**XS**
IF	—	0.9870	0.9727	**0.0194**	0.9989
RSH	**0.0135**	—	0.9542	$\mathbf{6.7540 \times 10^{-5}}$	0.9981
LODA	**0.0282**	0.0471	—	0.0003	0.9999
HST	0.9812	0.9999	0.9997	—	1.0000
XS	**0.0012**	**0.0020**	**0.0001**	$\mathbf{2.0490 \times 10^{-5}}$	—

Table 5: p-values from a Wilcoxon signed-rank test on the average precisions reported on all the datasets. Bold values are significant at $p = 0.05$. Each cell (i, j) contains the p-value of the hypothesis that method i has a greater AP than method j. It can be observed that **XS** > **LODA** > **RSH** > **IF** > **HST** is statistically significant at $p = 0.05$.

When repeated on just the high-dimensional datasets, we can conclude that **RS** > **LODA**, **RS** > **HST** and **IF** > **HST** at $p = 0.05$. We do not have enough evidence to make enough comparisons to construct a total order among methods.

	IF	HST	RSH	LODA	XS
IF		0.4839	0.9999	0.9977	1
HST	0.5321		0.9999	0.9299	1
RSH	0.0001307	0.0001049		0.0001974	1
LODA	0.002552	0.07526	0.9998		1
XS	1.91E-06	2.86E-06	9.54E-07	6.68E-06	

Table 6: Wilcoxon test p-value result for Low Dimension dataset. XST > RSH > LODA > HST $\sim$ IF

	IF	HST	RSH	LODA	XS
IF		0.006183	0.7432	0.07528	0.2853
HST	0.9944		0.9999	0.7996	0.7738
RSH	0.2689	0.0001156		0.02964	0.2729
LODA	0.9299	0.211	0.9728		0.835
XS	0.7271	0.2375	0.7392	0.1744	

Table 7: Wilcoxon test p-value result for High Dimension dataset. IF > HST, RSH > HST, RSH > LODA

	IF	HST	RSH	LODA	XS
IF		0.01942	0.987	0.9727	0.9989
HST	0.9812		0.999	0.997	1
RSH	0.0135	6.75E-05		0.9542	0.9981
LODA	0.0001096	0.02815	0.0003357	0.04711	0.9999
XS	0.001196	2.05E-05	0.002026	0.0001096	

Table 8: Wilcoxon test p-value result for all datasets. XS > LODA > RSH > IF > HST

Dataset	IF	HST	RSH	LODA	XS
High-dimensional Datasets					
gisette(30,0.3,1.2)	0.683 ± 0.045	0.46 ± 0.005	0.628 ± 0.021	0.531 ± 0.018	0.5282 ± 0.0091
gisette (30,0.3,10.0)	0.541 ± 0.027	0.354 ± 0.002	0.471 ± 0.024	0.321 ± 0.004	0.3204 ± 0.0029
gisette (30,0.3,20.0)	0.45 ± 0.022	0.308 ± 0.003	0.347 ± 0.017	0.29 ± 0.003	0.292 ± 0.0032
gisette (30,0.3,30.0)	0.432 ± 0.016	0.31 ± 0.002	0.316 ± 0.006	0.301 ± 0.005	0.2929 ± 0.0023
isolet (30,0.3,1.2)	0.537 ± 0.03	0.441 ± 0.01	0.559 ± 0.028	0.58 ± 0.03	0.6396 ± 0.0204
isolet (30,0.3,10.0)	0.372 ± 0.011	0.334 ± 0.004	0.391 ± 0.015	0.35 ± 0.007	0.3479 ± 0.0066
isolet (30,0.3,20.0)	0.33 ± 0.009	0.311 ± 0.001	0.322 ± 0.007	0.313 ± 0.006	0.3176 ± 0.0017
isolet (30,0.3,30.0)	0.3 ± 0.002	0.296 ± 0.001	0.293 ± 0.002	0.292 ± 0.004	0.2899 ± 0.0016
letter (30,0.3,1.2)	0.49 ± 0.026	0.44 ± 0.01	0.519 ± 0.053	0.534 ± 0.03	0.5967 ± 0.0204
letter (30,0.3,10.0)	0.374 ± 0.015	0.339 ± 0.008	0.375 ± 0.007	0.337 ± 0.003	0.3361 ± 0.005
letter (30,0.3,20.0)	0.309 ± 0.008	0.291 ± 0.002	0.31 ± 0.007	0.289 ± 0.002	0.3 ± 0.0015
letter (30,0.3,30.0)	0.319 ± 0.01	0.308 ± 0.001	0.307 ± 0.004	0.305 ± 0.003	0.3095 ± 0.0023
madelon (5,0.05,1.2)	0.896 ± 0.051	0.92 ± 0.007	0.969 ± 0.039	1.0 ± 0.0	1.0 ± 0.0
madelon (5,0.05,10.0)	0.643 ± 0.13	0.784 ± 0.068	0.899 ± 0.094	0.988 ± 0.012	0.9567 ± 0.023
madelon (5,0.05,20.0)	0.24 ± 0.093	0.16 ± 0.039	0.377 ± 0.143	0.112 ± 0.027	0.125 ± 0.0133
madelon (5,0.05,30.0)	0.07 ± 0.013	0.083 ± 0.004	0.099 ± 0.059	0.054 ± 0.007	0.0444 ± 0.0034
Low/medium-dimensional Datasets					
cancer (100,0.1)	0.661 ± 0.036	0.727 ± 0.01	0.702 ± 0.026	0.839 ± 0.024	0.8511 ± 0.0216
cancer (1000,0.1)	0.476 ± 0.043	0.394 ± 0.007	0.531 ± 0.047	0.686 ± 0.047	0.8453 ± 0.0179
cancer (2000,0.1)	0.447 ± 0.026	0.444 ± 0.015	0.474 ± 0.046	0.643 ± 0.066	0.8282 ± 0.0212
cancer (5000,0.1)	0.404 ± 0.021	0.4 ± 0.013	0.425 ± 0.022	0.508 ± 0.086	0.7729 ± 0.0394
ionosphere (100,0.1)	0.756 ± 0.031	0.773 ± 0.005	0.725 ± 0.018	0.771 ± 0.013	0.9005 ± 0.0052
ionosphere (1000,0.1)	0.59 ± 0.044	0.573 ± 0.008	0.579 ± 0.054	0.742 ± 0.032	0.8706 ± 0.01
ionosphere (2000,0.1)	0.441 ± 0.033	0.505 ± 0.033	0.446 ± 0.027	0.736 ± 0.029	0.8743 ± 0.0052
ionosphere (5000,0.1)	0.403 ± 0.022	0.376 ± 0.006	0.399 ± 0.028	0.675 ± 0.048	0.7966 ± 0.0117
telescope (100,0.1)	0.593 ± 0.01	0.576 ± 0.005	0.601 ± 0.015	0.617 ± 0.003	0.6373 ± 0.0077
telescope (1000,0.1)	0.451 ± 0.018	0.419 ± 0.008	0.471 ± 0.021	0.592 ± 0.013	0.6095 ± 0.0054
telescope (2000,0.1)	0.407 ± 0.021	0.407 ± 0.003	0.409 ± 0.018	0.586 ± 0.007	0.5872 ± 0.0087
telescope (5000,0.1)	0.376 ± 0.011	0.38 ± 0.002	0.378 ± 0.005	0.538 ± 0.026	0.5752 ± 0.0074
indians (100,0.1)	0.477 ± 0.013	0.444 ± 0.009	0.478 ± 0.007	0.472 ± 0.01	0.5218 ± 0.0077
indians (1000,0.1)	0.385 ± 0.017	0.379 ± 0.006	0.397 ± 0.024	0.444 ± 0.014	0.4976 ± 0.0106
indians (2000,0.1)	0.35 ± 0.014	0.361 ± 0.005	0.369 ± 0.023	0.41 ± 0.016	0.4777 ± 0.0082
indians (5000,0.1)	0.343 ± 0.008	0.338 ± 0.003	0.349 ± 0.014	0.4 ± 0.028	0.4659 ± 0.009

Table 9: Average precision of static methods on high-dimensional (top) and low/medium-dimensional (bottom) datasets. Mean and standard deviation reported over 10 runs. Numbers in the brackets indicate: (top) the percentage of features, fraction of samples to which noise is added, signal-to-noise ratio, and (bottom) noise column amount (as % of original dimensionality), relative noise factor.

3.5 Time and Space Complexity

Table 10: Symbol List

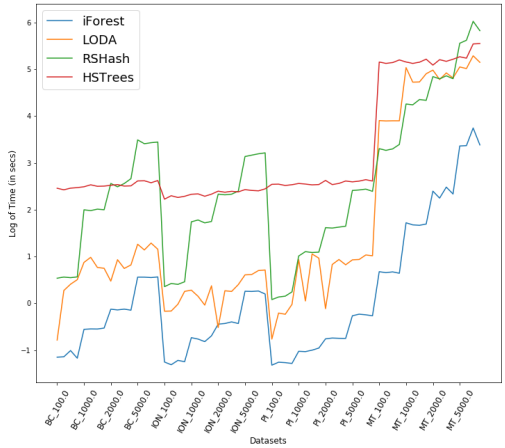
Symbols	Description
N	Number of data points
D	Number of dimensions/features
C	Number of ensemble components (trees, chains, histograms, etc.)
d	Max depth of trees/chains
ψ	iForest/HS-Trees/RS-Hash: sampling size
r	RS-Hash: Number of sampled dimensions
b	LODA: number of histogram bins
m	X-STREAM/RS-Hash: Number of CMS hash functions
L	X-STREAM/RS-Hash:: CMS hash table size
k	X-STREAM: Number of random projections
M	X-STREAM: Number of subspaces
W	X-STREAM: Number of windows

Table 11: Time and space complexity of the ensemble anomaly detection algorithms compared in this paper. In streaming case, a data point/vector arrives at a time for HS-Stream, LODA, and RS-Hash; whereas an update to a single feature for a data point arrives at a time.

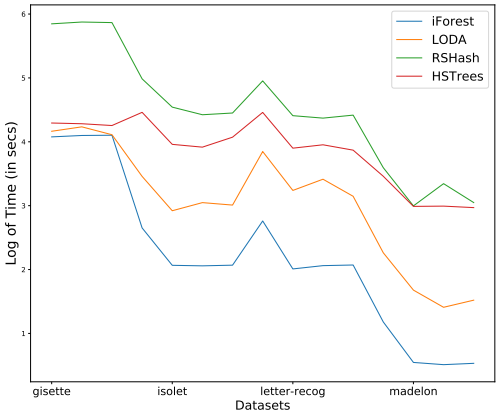
Algorithm	**Time Complexity**		**Space Complexity**
	Training	Scoring/Updating	
Batch/Offline			
iForest	$O(C\psi \log \psi)$	$O(C \log \psi)$	$O(C\psi)$
HS-Trees	$O(Cd\psi)$	$O(Cd)$	$O(C2^d)$
LODA	$O(NC\sqrt{D})$	$O(C\sqrt{D})$	$O(C\sqrt{D} + Cb)$
RS-Hash	$O(Crm\psi)$	$O(Crm)$	$O(CLm)$
X-STREAM	$O(M[NDk + Ckmd])$	$O(M[Dk + Ckmd])$	$O(MCLmd)$
Streaming/Online			
HS-Stream	–	$O(Cd + C\psi)$	$O(C2^d)$
LODA	–	$O(C\sqrt{D} + Cb)$	$O(C\sqrt{D} + Cb)$
RS-Hash	–	$O(Crm)$	$O(CLm)$
X-STREAM	–	$O(WMCkmd)$	$O(WMCLmd)$

3.6 Running time

iForest is the fastest, RSHash is the slowest (RSHash implementation, however can be optimized)



(a) Low/Mid - dimensional dataset analysis



(b) High dimensional dataset analysis

Figure 12: RunTimes of algorithms over datasets with different noise levels. X-axis is over different datasets with different noise levels. Y-axis is log of runtime (in seconds).

4 Dynamic evaluation

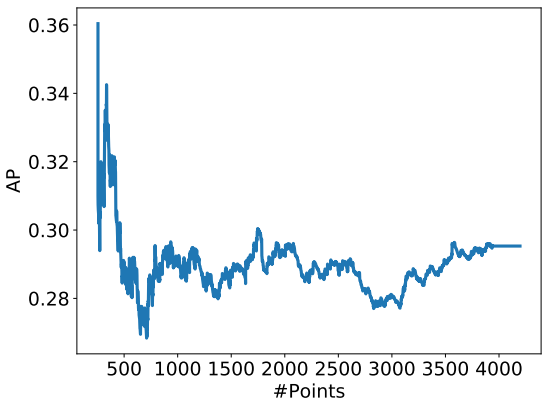
Datasets:

We use the same datasets as in HighDimension case in static setting. We create 10 shuffled versions for each dataset. The datasets could either be shuffled randomly or it could be clustered in time. For the clustered shuffled, the input also includes a clustering percentage, that is how much percentage of anomalies should be clustered in time. We currently do not experiment with clustered in time datasets.

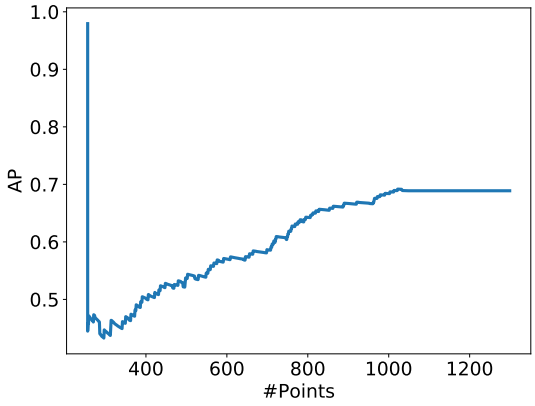
Algorithms

- RS-Hash: We use decay constant to be 0.015, and use first 256 samples to setup the 100 components, and then compute AUC/AP over time.
- HS-Trees:
- LODA:

Results



(a) Low/Mid - dimensional dataset analysis



(b) High dimensional dataset analysis

Figure 13: AP for two of the baseline datasets over time, using RS-Hash Streaming Algorithm.

References

Ahle, Thomas D et al. (2017). "Parameter-free locality sensitive hashing for spherical range reporting". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, pp. 239–256.

Andoni, Alexandr et al. (2017). "LSH forest: Practical algorithms made theoretical". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, pp. 67–78.

Aumasson, Jean-Philippe et al. (2012). "SipHash: A Fast Short-Input PRF." In: *INDOCRYPT*. Vol. 7668. Springer, pp. 489–508.

Bawa, Mayank et al. (2005). "LSH forest: self-tuning indexes for similarity search". In: *Proceedings of the 14th international conference on World Wide Web*. ACM, pp. 651–660.

Datar, Mayur et al. (2004). "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, pp. 253–262.

Li, Ping et al. (2006). "Very sparse random projections". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 287–296.