Fast Outlier Detection in Oblique Subspaces

Bowen Li
Florida State University
Tallahassee, Florida, USA
bli@cs.fsu.edu

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA
charu@us.ibm.com

Peixiang Zhao
Florida State University
Tallahassee, Florida, USA
zhao@cs.fsu.edu

Abstract

Subspace outlier detection is a fundamental data mining task for high-dimensional data with diverse applications across widely varying domains. Existing subspace hashing methods have been capable of identifying accurate and interpretable outliers in *axis-parallel* subspaces in linear time. However, these methods simply fail when applied to arbitrary-shaped, schema-less data that lack well-defined attributes or dimensions, such as time series and graphs. In this paper, we introduce a new notion of *oblique subspaces* defined on pairwise object proximity functions without requiring explicit multidimensional representations of the underlying data. By hashing data objects into arbitrarily oriented oblique subspaces, we can construct subspace hashing histograms for efficient and cost-effective outlier detection. Our proposed solution, OS-Hash (**O**blique **S**ubspace **Hash**ing), is a linear-time, constant-space method applicable to not only multidimensional but also arbitrary-shaped data, and it can further be extended to the data stream setting for subspace outlier detection. Our experimental studies on real-world multidimensional, time series, and graph data demonstrate the efficiency and efficacy of OS-Hash, which outperforms state-of-the-art outlier detection methods in terms of both runtime performance and accuracy.

CCS Concepts

• **Information systems → Data mining**.

Keywords

Outlier detection; Oblique subspaces; Data streams

ACM Reference Format:
Bowen Li, Charu C. Aggarwal, and Peixiang Zhao. 2025. Fast Outlier Detection in Oblique Subspaces. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM '25), November 10–14, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746252.3761354

1 Introduction

In high-dimensional datasets, an outlier is a data object deviating significantly from the general patterns of underlying data, often appearing distant or unusual to other objects (*a.k.a.* inliers) [4]. Outliers may result from intrinsic variability within data, inadvertent errors or distortions during data acquisition and entry, or represent rare, exceptional events with potentially valuable insights [2]. There have been numerous high-dimensional outlier detection methods devised in widely varying real-world applications, such as intrusion identification, medical diagnosis, financial fraud detection, and traffic management [7, 10, 20].

To date, the high dimensionality of data has posed a significant computational challenge to outlier detection due in particular to the existence of a multitude of irrelevant dimensions that may mask the intrinsic outliers to be identified [4]. It is thus essential to explore subspaces of multidimensional data for accurate and interpretable outlier detection. A proliferation of subspace outlier detection techniques have been proposed, including the original work [4], feature bagging [21], contrast-based ranking [17], statistical subspace selection [26, 27], and spectral methods [13, 31]. However, a majority of such methods are computationally expensive due to repeated subspace selections or costly eigenvector computation, rendering their time-complexity to be at least $O(n^2)$, which is impractical in real-world settings.

Recently, a subspace hashing method, RS-Hash [32], was proposed to identify outliers from hashing-based subspace histograms. Essentially, RS-Hash discretizes different value ranges of dimensions in order to create randomized multidimensional buckets. The count of objects in a subspace bucket is used to define the *outlierness* of objects (an outlier score with lower values indicative of a greater degree of outlierness). The main issue of RS-Hash is that it is inherently designed for multidimensional data with pre-defined dimensions or attributes, and as a result, the outliers identified are strictly confined to *axis-parallel* subspaces. This leads to a natural question: *How can the subspace hashing idea be used for arbitrary-shaped or schema-less data without explicit dimensions?*

In this paper, we provide a solid answer to the above question by proposing OS-Hash (**O**blique **S**ubspace **Hash**ing) that enables subspace hashing in arbitrarily oriented subspaces. OS-Hash supports subspace outlier detection for not only multidimensional but also any arbitrary-shaped objects, such as time series and graphs without readily available attribute or dimension information. In particular, we make the following contributions in OS-Hash:

(1) We introduce a new notion of *oblique subspace*, which breaks the barrier of axis-parallel subspaces in outlier detection. The oblique subspaces are defined for any type of data without explicit access to their vectorized multidimensional representations, as long as a pairwise proximity/distance function exists between the data objects (Section 2);
(2) We design the linear-time, constant-space method, OS-Hash, for outlier detection in oblique subspaces. This method is general enough to support outlier detection for both high-dimensional and arbitrary-shaped data without pre-defined attributes or dimensions (Section 3);

(3) We extend the idea of OS-Hash to the data stream setting, where outliers can be identified in real time from time-decayed data streams (Section 4);
(4) We perform experimental studies on real-world and synthetic time-series, graph, and multidimensional datasets and data streams, and the results demonstrate the effectiveness, efficiency, and generality of the proposed method, OS-Hash, against numerous outlier detection solutions, including the state-of-the-art method, RS-Hash (Section 5).

2 Problem Setting and Oblique Subspaces

We consider a data collection $O = \{O_1, O_2, \ldots, O_n\}$ consisting of n data objects, which can be of any data type, such as multidimensional vectors conforming to predefined object attributes or dimensions, or schema-less time series, graphs, or free text without explicit dimension information. The only premise underlying our problem setting is an existence of a pairwise object proximity function, $s : O \times O \to \mathbb{R}_{\geq 0}$, that quantifies the pairwise similarity between objects of O. Specifically, given any pair of data objects O_i and O_j of O, their similarity is denoted $s_{ij} = s(O_i, O_j)$. We ensure an on-the-fly computation of s_{ij}, whereas it is impractical to compute upfront all $O(n^2)$ pairwise similarities in O, especially for large real-world datasets.

We consider the *subspace hashing* framework for outlier detection, which scales linearly with the dataset size, and provides accurate and interpretable subspace outliers for high-dimensional datasets [2, 32]. This approach starts with a sample $\mathcal{S} \subseteq O$ that is hashed to maintain histograms as a trained model for objects of $\mathcal{S}$ within different randomized bounding regions of localized, axis-parallel subspaces. Thereafter, all data objects of O are hashed the same way to these subspaces and get scored *w.r.t.* the trained model. Objects mapped to a histogram with extremely low scores are identified as outliers in the low-dimensional subspaces. Remark that this approach assumes object dimensions pre-specified for O, which, however, may not hold for arbitrary-shaped data. Although it is possible, in principle, to generate multidimensional embeddings of O based on similarity-preserving kernel methods [16, 22], it typically requires $O(n^3)$ time, which is too costly in practice.

In this paper, we identify *oblique subspaces* of O based on the pairwise object proximity *without* requiring explicit access to the multidimensional representations of O. For the ease of exposition, we assume, for each data object $O_i \in O$, $1 \leq i \leq n$, its multidimensional representation is $\overrightarrow{X_i}$. To clarify, this does not mean that we should explicitly know this vector representation or its dimensionality; it is just a theoretical assumption that can be relaxed in subsequent analysis. Given two objects O_i and O_j of O, we assume the dot-product similarity between $\overrightarrow{X_i}$ and $\overrightarrow{X_j}$ indicates their object proximity, s_{ij}:

$$s_{ij} = \overrightarrow{X_i} \cdot \overrightarrow{X_j}. \tag{1}$$

In practice, however, s_{ij} can be any similarity function for the actual data type of O. For instance, for time series data, we can use the dynamic time warping (DTW) similarity [18, 25]; for graphs, we may choose graph-kernel based similarities [28, 36]. If O is already multidimensional with well-defined attributes or dimensions, a direct application of the dot-product similarity or a Euclidean-based similarity just suffices.

Our idea to identify oblique subspaces of O is that, instead of treating each data object independently, we consider *a pair of* objects from O to construct an *oblique* vector direction, along which all objects of O can be projected based on the object proximity function, s. By repeatedly picking r pairs of objects that identify an r-dimensional oblique subspace of O, we can create, as a consequence, a histogram in this oblique subspace for outlier detection. To be specific, we select two object O_i and O_j at random from O, and project all the objects of O along the oblique direction from O_i to O_j. In this case, O_i obtains a coordinate of zero, whereas O_j obtains a positive coordinate. For the rest of the objects $O_k \in O$, where $k \neq i, j$, its projection, denoted $\mathbf{proj}(O_k)$, can be computed as the dot-product of $(\overrightarrow{X_k} - \overrightarrow{X_i})$ with the oblique direction $(\overrightarrow{X_j} - \overrightarrow{X_i})$:

$$\begin{aligned} \mathbf{proj}(O_k) &= (\overrightarrow{X_k} - \overrightarrow{X_i}) \cdot (\overrightarrow{X_j} - \overrightarrow{X_i}) \\ &= \overrightarrow{X_k} \cdot \overrightarrow{X_j} - \overrightarrow{X_k} \cdot \overrightarrow{X_i} - \overrightarrow{X_i} \cdot \overrightarrow{X_j} + \overrightarrow{X_i} \cdot \overrightarrow{X_i} \\ &= s_{kj} - s_{ik} - s_{ij} + s_{ii}. \end{aligned} \tag{2}$$

Note that the projection, $\mathbf{proj}(O_k)$, along the oblique direction determined by O_i and O_j is dependent only on the object proximity computation as presented in Equation 2, and it is completely unnecessary to access the vectorized, multidimensional representations of O. From this perspective, the idea of oblique subspaces shares a conceptual resemblance to the kernel methods [33, 35].

3 OS-Hash

In this section, we discuss our oblique-subspace based solution, OS-Hash. OS-Hash is inherently an ensemble approach that manages a combination of m base detectors for subspace outlier detection. First, we consider a training sample $\mathcal{S} \subseteq O$ to create a *subspace histogram model* for each base detector in a randomized r-dimensional oblique subspace. Next, every object $O_i \in O$ is projected along the oblique dimensions and further hashed and scored within different bounding regions of oblique subspaces. To manage the histogram models efficiently, we adopt the Count-Min sketch [12, 39], where w pairwise independent hash tables are maintained for accurate estimation of outlier scores with theoretical performance guarantees. As a consequence, individual outlier scores across m different base detectors are further synthesized in order to achieve extraordinarily accurate and robust outlier detection performance. For the ease of reference, we provide a primer of notations for OS-Hash in Table 1.

In the following, we present the execution steps for an individual base detector of OS-Hash:

1. **Parameter Selection.** Determine appropriate values for the locality parameter $f \in (0, 1)$ and the oblique subspace dimensionality r for the current base detector. Remark that f defines the histogram level in terms of the fractional width of the buckets along each oblique dimension. The dimensionality r is also related to the width of buckets: a lower dimensionality leads to narrower buckets as we tend to ensure a sufficiently expected number of objects in the buckets. Both parameters are randomized and vary across different base detectors. The method for selecting appropriate values for these parameters is discussed in Section 3.2.

2. **Oblique Subspace Identification.** Sample r pairs of objects at random from O. The ith pair of objects, (O_{a_i}, O_{b_i}), where $1 \leq a_i \neq$

Table 1: A Primer of Notations

Notation	Explanation
O	A collection of n data objects: $O = \{O_1, \ldots, O_n\}$
$s_{ij}, s(O_i, O_j)$	Object similarity between O_i and O_j
$\mathcal{S} \subseteq O$	A object sample with $\lvert\mathcal{S}\rvert = s = \min\{n, 1000\}$
m	Number of base detectors in OS-Hash
f	Locality parameter $f \in (1/\sqrt{s}, 1 - 1/\sqrt{s})$ *u.a.r.*
r	Oblique subspace dimensionality (Equation 5)
$\alpha_i (1 \le i \le r)$	Shift parameter in dimension i, $\alpha_i \in (0, f)$ *u.a.r.*
z_{ij} or z'_{ij}	Projected value for jth dimension of O_i
$\overrightarrow{Z'_i} = (z'_{i1}, \ldots, z'_{ir})$	Normalized r-dimensional representation of O_i
$\overrightarrow{Y_i} = (y_{i1}, \ldots, y_{ir})$	r-dimensional bucket representation of O_i
$\mathcal{H}$	The CountMin sketch
w	Number of hash tables in $\mathcal{H}$
l	Size of each hash table in $\mathcal{H}$: $l = 10 * s$
es^i_j	Estimated count of O_i in jth base detector
os^i_j	Outlierness of O_i in jth base detector

$b_i \le n$ and $1 \le i \le r$, is used to determine the ith oblique dimension. As a result, r-dimensional oblique subspaces are identified for O.

3. **Sample Data Projection.** Select a sample $\mathcal{S} \subseteq O$ of objects at random, where $|\mathcal{S}| = s$, and project the objects of $\mathcal{S}$ along each of r oblique dimensions specified by (O_{a_i}, O_{b_i}) according to Equation 2. To this end, each sample object $O_i \in \mathcal{S}$ is represented as an r-dimensional vector in the oblique subspace with the value of the jth oblique dimension denoted by z_{ij}, where $1 \le i \le s$ and $1 \le j \le r$. We further normalize z_{ij} to z'_{ij} based on the min-max normalization:

$$z'_{ij} \Leftarrow \frac{z_{ij} - \min_j}{\max_j - \min_j}, \tag{3}$$

where $\min_j$ and $\max_j$ are the minimum and maximum values along each oblique dimension j in the sample $\mathcal{S}$, respectively. Each sample object O_i is thus converted to a normalized r-dimensional vector $\overrightarrow{Z'_i} = (z'_{i1}, \ldots, z'_{ir})$. We further create for each O_i a new r-dimensional discrete vector $\overrightarrow{Y_i}$ that contains the integer bucket values of O_i using the fractional width f for each bucket. Specifically, we set y_{ij} with the integer value of $\lfloor (z'_{ij} + \alpha_j)/f \rfloor$, where α_j is a shift parameter drawn uniformly at random from $(0, f)$. Note here we introduce an r-dimensional random vector $(\alpha_1, \ldots, \alpha_r)$, with each shift parameter α_j fixed across all sample objects of $\mathcal{S}$ for the dimension j. It is designed to address the edge effects in the first and last buckets of the histogram, and also offer diversity to the whole ensemble. In essence, the formulation that defines bounding regions is $\lfloor (z'_{ij} + \alpha_j)/f \rfloor$, which maps all objects within a particular range along the oblique dimension j into a single bucket. A *histogram* is henceforth defined by a combination of r buckets representing all the r oblique dimensions. To this end, a subset of sample objects could be mapped to the same combination of r-dimensional histogram buckets, if the values of $\lfloor (z'_{ij} + \alpha_j)/f \rfloor$ are the same across *all* oblique dimensions j for such objects (indexed by i), and the counts of these r-dimensional histograms are used to quantify the outlierness of data objects.

4. **Oblique Subspace Hashing.** Construct a CountMin sketch, $\mathcal{H}$, for oblique subspace hashing and histogram maintenance. This sketch consists of w hash tables implementing w pairwise independent hash functions $\mathcal{H}_k : \mathbb{N}^r_{\ge 0} \to \mathbb{N}_{\ge 0}$, where $1 \le k \le w$, each of which takes as input the r-dimensional bucket vector $\overrightarrow{Y_i}$, and maps it to an integer value in the range of $(0, l-1)$, where l indicates the number of elements of these hash tables. After applying each hash function $\mathcal{H}_k$ upon $\overrightarrow{Y_i}$, we increment the value of $\mathcal{H}_k(\overrightarrow{Y_i})$ by 1, where $1 \le k \le w$.

5. **Outlier Score Evaluation.** Apply the oblique subspace histogram model built from the sample $\mathcal{S}$ toward the whole dataset O in order to compute an outlier score for every object of O. In particular, we transform each object $O_i \in O$ $(1 \le i \le n)$ into its r-dimensional bucket representation $\overrightarrow{Y_i}$ the same way as we discussed in Step 3 for sample objects of $\mathcal{S}$. The values of $\min_j$ and $\max_j$ $(1 \le j \le r)$, which are derived from the sample $\mathcal{S}$ along r-dimensional oblique subspaces, can be used as an approximation for the whole dataset O. We then insert each $\overrightarrow{Y_i}$ into the CountMin sketch, $\mathcal{H}$, by consecutively applying w hash functions, $\mathcal{H}_1, \ldots, \mathcal{H}_w$ upon $\overrightarrow{Y_i}$. Let the value of the $\mathcal{H}_k(\overrightarrow{Y_i})$-th cell in the kth hash table be c_k that corresponds to the count of the cell, where $1 \le k \le w$. Note that c_k could be zero when an object O_i is not included in the sample $\mathcal{S}$ (or c_k is at least 1 otherwise). As a result,

- if O_i was originally in the sample $\mathcal{S}$, we report its outlier score as $\log_2(\min\{c_1, \ldots, c_w\})$;
- otherwise, report $\log_2(\min\{c_1, \ldots, c_w\} + 1)$ as O_i's outlier score.

The use of logarithm above is intended to create a similar interpretation as a log-likelihood fit (cf. Section 3.1). Note that a lower outlier score implies that the object O_i falls in histogram buckets with sparser counts or a lower aggregated log-likelihood fit, and consequently, a greater tendency for it to be an outlier.

The discussion above is about the construction of, and outlier estimation from, a single base detector of OS-Hash. This process can be repeated m times, once for each base detector of the ensemble, and the outlier scores of all m ensemble components are further averaged as the final outlier score for each object of O. Formally, let os^i_j represents the outlier score of the ith object from the jth base detector, where $1 \le i \le n$ and $1 \le j \le m$. The final outlier score of the object O_i based on OS-Hash is

$$\text{OS-Hash}(O_i) = \tfrac{1}{m} \textstyle\sum_{j=1}^{m} os^i_j. \tag{4}$$

Note that each ensemble component is a relatively weak outlier detector because it uses the density of a bucket region in a randomized subspace. However, OS-Hash has proven to be quite accurate due in particular to its averaging effect of multiple independent ensemble components [3, 43].

In summary, OS-Hash creates sparse bounding regions along a set of (virtual) dimensions that characterize an oblique subspace for the multidimensional embedding of data. Interestingly, when the objects are multidimensional with pre-defined attributes or dimensions and the object similarity function is dot-product, this bounding region is not just a theoretical construct, but can be identified within the actual multidimensional space. The reason to consider a sampling approach in OS-Hash is to ensure greater diversity over different base detectors. Another diversity of OS-Hash stems from the fact that values of f and r are chosen in a randomized way. It is also important to note that the inclusion of the random vector, $(\alpha_1, \ldots, \alpha_r)$, in Step 3 is crucial for making OS-Hash work: these randomized parameters ensure that even when the same oblique dimension is sampled in different base detectors, the region boundaries tested for a given data object can be different. This provides even more diversity to test different subspace localities of data in different ensemble components.

3.1 Log-Likelihood Interpretation

We denote by ec^i_j the *estimated count* of object O_i from the base detector j, which is equivalent to $\min\{c_1, \ldots, c_w\}$ derived from the CountMin sketch. Assume O_i is not in the sample S, then its likelihood within one particular base detector is ec^i_j/s, the probability that an object falls in a bucket with ec^i_j objects out of $s = |S|$ sample objects. Likewise, its likelihood over all m base detectors of OS-Hash can be estimated by $\prod_{j=1}^{m}(ec^i_j/s)$. The corresponding log-likelihood of the object O_i, $L(O_i)$, is thus defined as follows:

$$L(O_i) = \log(\textstyle\prod_{j=1}^{m}(ec^i_j/s)) = \sum_{j=1}^{m}\log(ec^i_j) - m\log s = \sum_{j=1}^{m} os^i_j + const,$$

where os^i_j represents the outlier score of O_i from the jth base detector, and the term $(-m\log s)$ is a constant. Therefore, the estimated log-likelihood fit is directly related to the outlier score, os^i_j, as computed at Step 5 of the OS-Hash algorithm.

3.2 Parameter Analysis

There are a few algorithmic parameters for OS-Hash as follows:

Sample Size s. The sample size is typically a small constant, making OS-Hash cost-effective especially for large datasets. As the sample $\mathcal{S}$ cannot be larger than the dataset O, we often set $s = \min\{n, 1000\}$ empirically.

Locality parameter f. The locality parameter defines the width of the buckets of a histogram, and its value is drawn uniformly at random in the range of $(1/\sqrt{s}, 1 - 1/\sqrt{s})$, where s is the sample size. Since f corresponds to the *relative* width of buckets as a fraction of the entire range of data, by choosing different values of f at random in the aforementioned range, it ensures that the histograms of OS-Hash have varying levels of granularity.

Oblique Subspace Dimensionality r. The dimensionality r is chosen as an integer uniformly at random in the range of

$$\left[1 + 0.5\lfloor \log_{\max\{2,1/f\}}(s)\rfloor, \lceil \log_{\max\{2,1/f\}}(s)\rceil\right]. \quad (5)$$

Since $1 + 0.5\lfloor \log_{\max\{2,1/f\}}(s)\rfloor \geq 1 + 0.5\lfloor \log_{\sqrt{s}}(s)\rfloor = 2$, the lower bound of r is at least 2. The above inequality also explains why the lower bound of f is set $1/\sqrt{s}$. Remark that the expected number of objects in a histogram region lies between 1 and $\sqrt{s}$ depending on the specific values of the dimensionality r. As a result, the variations in the local density of histograms can be regulated precisely, making the oblique subspace regions not too sparse nor too dense. Additionally, the variability across different base detectors ensures that the ensemble approach can identify outliers accurately after averaging among m base detectors [3].

3.3 Count-Min Sketch Based Hashing

In OS-Hash, we employ the CountMin sketch [12], $\mathcal{H}$, for oblique subspace hashing. $\mathcal{H}$ contains w hash tables, each of which implements a pairwise independent hash function that maps an r-dimensional bucket vector $\overrightarrow{Y}$ to an integer value in the range of $(0, l-1)$, where l is the number of entries in hash tables. By setting $l = 10 * s$, we witness very rare collisions even on a single hash table. In practice, we set $w = 4$ and $l = 10,000$, making the sketch smaller than 200KB, which is very space-efficient even in numerous space-constrained settings.

We denote by $\mathcal{H}_1(\overrightarrow{Y}), \ldots, \mathcal{H}_w(\overrightarrow{Y})$ the corresponding counts of $\overrightarrow{Y}$ in w hash tables of the sketch $\mathcal{H}$, respectively. Note that $c_k = \mathcal{H}_k(\overrightarrow{Y})$ is an overestimate of the actual count of $\overrightarrow{Y}$ because of hash collisions, where $1 \leq k \leq w$. Likewise, the minimum value, $\min\{c_1, \ldots, c_w\}$, is still an overestimate. However, this estimation error in the CountMin sketch has proven to be significantly small as shown below.

Consider a sample object falling in a specific cell of a hash table $\mathcal{H}_k$, which can be modeled as a Bernoulli random variable. Since $\mathcal{H}_k$ maps to a range of length l, the success probability for this random variable is $1/l$, Given any object $O_i \in O$, the probability that O_i collides with any of the s samples in $\mathcal{H}_k$ is also $1/l$, and the probability of no collisions between O_i and all s samples is therefore $(1 - 1/l)^s$. As there exist w pairwise independent hash tables in the CountMin sketch, the probability that O_i has collisions within all w hash tables is $(1 - (1 - 1/l)^s)^w$. According to the principle of the CountMin sketch [12], it provides an exact estimation when no collisions arise in *at least* one of the w hash tables, the probability of which is $1 - [1 - (1 - 1/l)^s]^w$. Given typical parameter settings for OS-Hash as $s = 1000$, $l = 10 * s = 10,000$, and $w = 4$, the above probability of exact estimation is approximately 0.9999. In other words, a single base detector can provide correct outlier scores with a high probability of 0.9999. An inaccuracy may arise due to a collision in any base detector. Over $m = 100$ base detectors, for instance, the probability of correct outlier score estimation drops to $0.9999^{100} \approx 0.99$, which is still high. Note that we can further boost this probability by incorporating more hash tables in $\mathcal{H}$ (*i.e.*, by increasing the value of w) without incurring notable space overhead for the sketch structure.

3.4 Complexity Analysis

In OS-Hash, a fundamental operation is the pairwise similarity computation, the objective of which is to create multidimensional representations for data objects in oblique subspaces. We denote by $O(T)$ the time complexity for the object similarity computation, which varies by particular object types and applications. For instance, if the data are already multidimensional and the proximity function is the dot product, then $T = O(d)$, where d is the dimensionality of O; When the data are time series and the proximity function is dynamic time warping, then $T = O(N^2/\log\log N)$ where N is the length of time series [15]; When the data are graphs, we typically choose linear-time vertex/edge histogram kernels, or polynomial-time Weisfeiler-Lehman (WL) kernel [34, 36, 40] for graph similarity computation.

For each base detector of OS-Hash, $O(nr)$ similarity computations are required, where n is the dataset size and r is the dimensionality of oblique subspaces. Since r is determined by the sample size in the form of $O(\log s)$ (cf. Section 3.2), the number of similarity computations in each base detector is thus $O(n\log s)$. For all m base outlier detectors, the time complexity of OS-Hash is $O(nmT\log s)$, making it a linear-time algorithm *w.r.t.* the dataset size n. Note the parameter m, the number of base outlier detectors, is a small constant typically ranging from 10 to 100.

The space complexity of OS-Hash is primarily related to the overall space overhead of the CountMin sketch, which is $O(w \cdot l)$, where w is the number of hash tables in the sketch and l is the size of each hash table. In typical settings, both w and l are constants, making the space complexity of OS-Hash constant as well.

4 OS-Hash in Data Streams

Another merit of OS-Hash is its extensibility to data streams, where a continuous and rapid flow of data objects arrive in real time [1, 14]. Unlike traditional datasets in the static setting, data streams are typically processed in one pass due in particular to their high volume, velocity, and transiency. Because of the rapid changes of underlying patterns in data streams, outliers within data streams are sensitive to such changes over time, making real-time outlier detection extremely challenging [24, 41]. Specifically, to account for the timeliness of streaming data, we often consider two time-evolving discounting options for outlier detection in data streams:

(1) We identify outliers within a *sliding window* of data objects, like within one minute, one hour, or one day of streaming data;
(2) We rely on a log-likelihood density model based on the time-decayed principle for streaming data.

It is straightforward to extend OS-Hash to the sliding-window setting, because the CountMin sketch readily admits object insertion and deletion: Incoming objects are automatically inserted to the sketch, and obsolete ones falling off from the sliding window are removed. The performance of this approach, as a result, is similar to the static case. In what follows, we primarily focus on the more difficult time-decayed setting.

The time-decayed model uses an exponential function with a decay rate λ to quantify the time-varying *weight* of an object O_i based on the number of objects that have arrived subsequently after O_i. Specifically, if t objects have arrived after O_i, the weight of O_i is $2^{-\lambda t}$. As such, the arrival of new objects will reduce the weights of earlier ones exponentially, and the parameter λ controls this decay rate. We thus make the following key modifications to OS-Hash in the case of time-decayed data streams:

- In the static setting, each base detector is created consecutively by repeated sampling of the original dataset, $\mathcal{O}$. This is impossible for the case of data streams. We thus maintain oblique subspace histograms of all ensemble components in the same CountMin sketch. In general, it requires a larger sketch to avoid potential collisions.
- The values of $\min_j$ and $\max_j$ are specific to each base detector in the static setting, whereas in the data stream, they can be estimated from an initial sample of streaming data. Although these values may change, they do not need to be exact in our approach to work.
- In the static setting, the number of samples for each base detector is $|\mathcal{S}| = s$. In data streams, however, older objects may weigh less than 1. We thus need to find the *weighted* number of samples for each base detector. Since each object contributes to each base detector in an exponentially decayed way, the total weight is $\sum_i 2^{-\lambda i} = 1/(1-2^{-\lambda})$. The *effective* sample size in data streams is thus defined as $s = \max\{1000, 1/(1-2^{-\lambda})\}$.
- In the static setting, the values of the locality parameter f are set differently for each base detector. In data streams, however, all base detectors are constructed simultaneously. Therefore, the values of locality parameters, $f_1, \ldots, f_m$, of m base detectors are sampled upfront in the range of $(1/\sqrt{s}, 1-1/\sqrt{s})$, where s is the weighted sample size discussed above.
- For each base detector i ($1 \leq i \leq m$), its oblique subspace is created upfront by sampling r_i pairs of objects. Here the dimensionality r_i is set according to Equation 5, where the weighted value of s is used instead.
- For jth oblique dimension of the ith base detector, where $1 \leq i \leq m$ and $1 \leq j \leq r_i$, the shift parameter α_{ij} is selected uniformly at random from $(0, f_i)$ and stored upfront. The resultant random vector $(\alpha_{i1}, \ldots, \alpha_{ir_i})$ is used consistently to determine the identity of relevant buckets for upcoming streaming objects at the time of model construction.

A critical issue in the time-decayed model is that the weight of each existing object has to be multiplied by a factor of $2^{-\lambda}$ on the arrival of every new object in the stream. Doing so explicitly would be too costly. We instead consider a *lazy* approach that maintains the *weighted* object counts implicitly in the CountMin sketch. In particular, for each sketch cell, besides the count, the time-stamp t_l of the *last* time it is updated is also maintained, and decaying the count of this sketch cell is done only when it is updated due to an increment of the count. Specifically, let $t_c > t_l$ be the *current* time-stamp at which a new object is streaming in. When a particular sketch cell needs to be updated, we first multiply the existing count of this cell by $2^{-\lambda(t_c - t_l)}$, and then increment it by 1. Additionally, the time-stamp of this sketch cell is updated to t_c. As such, the decayed weights of sample objects are updated in a lazy fashion in the CountMin sketch. At the step of outlier score evaluation, when a sketch cell is accessed, its count is multiplied by $2^{-\lambda(t_c - t_l)}$ before being reported, while both the count and time-stamp need no explicit update.

Once the key parameters of OS-Hash have been determined from an initial sample of the data stream, we consider an online approach for subspace outlier detection. In particular, when a streaming object O arrives, it is first projected into the oblique subspaces, and then normalized according to Equation 3. Note that both the training and test phases need to be executed simultaneously in the streaming setting: We first compute O's outlier score based on the current state of the CountMin sketch (the test phase) and then update the counts and time-stamps for the correspoonding cells of the sketch (the training phase):

(1) For each base detector $i \in \{1, \ldots, m\}$ and an oblique dimension $j \in \{1, \ldots, r_i\}$, compute the normalized representation z_{ij} for the streaming object O. The histogram bucket for the jth oblique dimension is $\lfloor (z_{ij} + \alpha_{ij})/f_i \rfloor$. As a result, the r_i-dimensional bucket representation $\overrightarrow{Y_i}$ for ith base detector includes r_i values of $\lfloor (z_{ij} + \alpha_{ij})/f_i \rfloor$ in the oblique subspace.
(2) For each base detector $i \in \{1, \ldots, m\}$, and each hash function $\mathcal{H}_k (1 \leq k \leq w)$ in the CountMin sketch, compute $\mathcal{H}_k(\overrightarrow{Y_i})$ in order to access the weighted count, denoted by c_k^i, from the corresponding cell of the CountMin sketch. The score for ith base detector is thus $\log(1 + \min\{c_1^i, \ldots, c_w^i\})$. Such scores from all m base detectors are further averaged as the ultimate outlier score of the streaming object O.

Table 2: General statistics of datasets.

Dataset	#Objs	#Dims	Outliers (%)
Static DataSets			
LYMPHOGRAPHY	148	18	3.4
CARDIO	1,831	21	9.6
MUSK	3,062	166	3.1
WAVEFORM	3,509	21	4.7
KDDCUP99	25,000	41	0.7
Streaming DataSets			
ACTIVITY	21,383	51	10.0
KDDCUP99-T	25,000	41	0.7

(a) Multidimensional Datasets

Dataset	#Objs	Outliers (%)
Static DataSets		
PICKUP	45	14.29
PEBBLE	120	12.50
POWER	600	14.00
ECG5000	3,039	3.94
CROP	16,500	2.42
Streaming DataSets		
ACTIVITY-T	21,383	10.0
CROP-T	16,500	2.42

(b) Time Series Datasets

Datasets	Graphs	Avg. $\|V\|$	Avg. $\|E\|$	Outliers (%)
Static Datasets				
MUTAG	135	19.24	21.76	7.4
FINGER	534	5.84	4.72	3.18
AIDS	1800	13.11	13.37	11.11
MUTAGEN	2500	29.66	30.54	3.96
TOX21	10000	18.41	18.87	3.89
Streaming Datasets				
TOX21-AR-T	10000	18.41	18.87	3.89
MCF-7-T	20000	27.43	29.68	10.00

(c) Graph Datasets

(3) Update both the counts and time-stamps in each of the w cells of the CountMin sketch based on the aforementioned time-decayed model and the lazy weighting strategy.

This outlier detection algorithm in data streams is referred to as OS-Stream, which continuously reports outlier scores of objects as they arrive, and then updates the decayed streaming model for future outlier detection. OS-Stream has proven to be extremely cost-effective as there are only a small number of operations incurred for each streaming object of the data streams.

5 Experiments

We report in this section the experimental studies for our proposed solutions, OS-Hash and OS-Stream, for subspace outlier detection. To demonstrate their generality, effectiveness, and efficiency, we consider both multidimensional and arbitrary-shaped data including time series and graphs. We evaluate OS-Hash and OS-Stream against state-of-the-art subspace outlier detection methods. All algorithms were implemented in C++, and complied by g++ 11.4.0 with the -O3 optimization option enabled. The experiments were performed on a Linux machine running Ubuntu Server 22.04.5 LTS with two Intel 2.3GHz ten-core CPUs and 256GB memory. The complete source code and data are publicly available online[1].

5.1 Datasets

We consider both real and synthetic datasets in the experiments, and label outliers based on the commonly used principles in the literature [2]: There are class labels for data objects in each dataset, and we use specific labels to categorize objects as inliers and outliers, provided that inliers and outliers are distinguishable by their pairwise distances. In particular, for the datasets containing rare classes, we label the objects in rare classes as outliers. If class distributions are roughly balanced, we downsample one class significantly and use it as the outlier label; The objects belonging to non-downsampled classes are thus inliers. The general statistics of all datasets, including multidimensional, time-series, and graph data, are reported in Table 2.

Multidimensional Datasets. We select five multidimensional data sets: LYMPHOGRAPHY, CARDIO, MUSK, WAVEFORM, KDDCUP99, and two streaming datasets: ACTIVITY, KDDCUP99-T, from the UCI Machine Learning Repository[2], which have been examined by RS-Hash [32] for subspace outlier detection.

Time Series Datasets. We consider five time series datasets: PICKUP, PEBBLE, POWER, ECG5000, CROP, and two streaming time-series datasets: ACTIVITY-T, CROP-T from the UCR Time Series Classification Archives[3].

Graph Datasets. We select five graph datasets: MUTAG, FINGER, AIDS, MUTAGEN, TOX21, and two streaming graph datasets: TOX21-AR-T, MCF-7-T, from TUDatasets[4], which maintains benchmark databases for graph classification and regression.

5.2 Evaluation Metrics

To evaluate the accuracy of outlier detection methods, we consider the Area Under the Curve (AUC) of the Receiver Operating Characteristics (ROC) curve, a standard metric to quantify the classification performance of binary classifiers that differentiate outliers from inliers in a given dataset. This effectiveness metric is used in both static and streaming settings. To evaluate algorithm efficiency, we consider the overall runtime (in seconds) of outlier detection algorithms in the static setting. In data streams, we instead consider the algorithm throughput, the number of objects processed per second, to quantify algorithm efficiency.

5.3 Baseline Methods

We compare our proposed solutions against a wide range of subspace outlier detection methods. In the static setting, we consider seven baseline methods in multidimensional datasets, including AvgKNN [7], FastABOD [20], iForest [23], HiCS [17], LOF [11], and RS-Hash [32]. LOF and HiCS are density-based outlier detection methods. AvgKDD is a kNN-based approach for outlier detection. FastABOD is an angle-based outlier detection method. iForest is a partition-based outlier detection algorithm. RS-Hash is the state-of-the-art subspace hashing method for outlier detection, and thus the main competitor of our solution, OS-Hash.

For distance-based solutions, including OS-Hash and RS-Hash, we consider the following pairwise object proximity/distance functions if not specified otherwise: the Euclidean distance for multidimensional datasets, the dynamic time warping (DTW) distance for time series, and the Weisfeiler-Lehman graph kernel distance

[1]https://github.com/BowenLi1994/OSHash

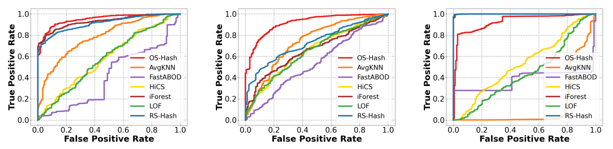
[2]https://archive.ics.uci.edu/datasets

[3]https://www.cs.ucr.edu/~eamonn/time_series_data

[4]chrsmrrs.github.io/datasets

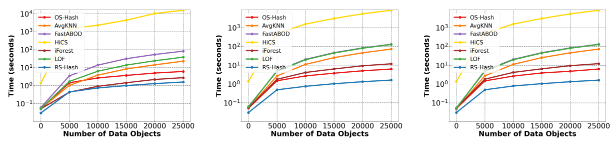
Table 3: AUC results for multidimensional datasets.

Datasets	OS-Hash	RS-Hash	AvgKNN	LOF	iForest	HiCS	FastABOD
LYMPHOGRAPHY	97.28	**99.92**	97.89	97.41	99.30	95.85	46.36
CARDIO	**94.98**	91.19	78.53	58	93.07	58.27	41.16
MUSK	**100**	100	24.10	39.17	100	39.50	48.78
WAVEFORM	**91.23**	72.97	73.83	65.03	66.20	65.23	53.68
KDDCUP99	92.91	**99.96**	14.35	44.69	99.94	52.19	38.27



(a) CARDIO (b) WAVEFORM (c) KDDCUP99

Figure 1: ROC curves in multidimensional datasets.



(a) KDDCUP99 (b) NORMAL(0,1) (c) UNIFORM(0,1)

Figure 2: Runtime costs in multidimensional datasets.

for graphs. We set the algorithm parameters of each method with the best possible values as reported in the literature. For instance, we set $k = 10$ for kNN search in LOF, AvgKNN, FastABOD, and HiCS. For ensemble-based solutions, RS-Hash and OS-Hash, we consider $m = 100$ base detectors in each method. For CountMin sketches, we use $w = 4$ pair-wise independent hash functions, each of which maps to $l = 10,000$ hash table entries by default. While evaluating arbitrary-shaped data such as time series and graphs, there exist no pre-specified dimensions or attributes. Therefore, only LOF, AvgKDD, and our proposed solution, OS-Hash, work in this scenario. We further introduce two more baselines: COF [38] and LoOP [19], both of which are localized density-based outlier detection methods applicable to time series and graphs.

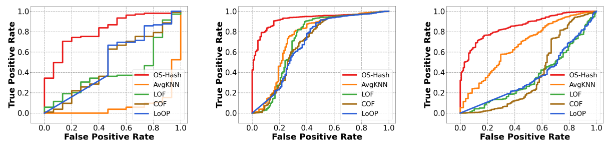
In the data streams setting, only RS-Hash, LOF, and AvgKNN work, and we denote their corresponding streaming variants as RS-Stream, LOF-Stream, and AvgKNN-Stream. We set the decay factor $\lambda = 0.015$, and when the decayed weight of an object is less than 10^{-5}, it is removed from the sketches without further consideration. This setup helps improve the accuracy for outlier detection when faced with concept drifts in fast evolving data streams. Remark that RS-Stream can only work in multidimensional data streams. In contrast, our method, OS-Stream, enables outlier detection for all different data types, including arbitrary-shaped time series and graphs in the streaming setting.

5.4 Results in The Static Setting

We first report experimental results of seven outlier detection methods on five real-world multidimensional datasets, and the AUC scores are reported in Table 3. Additionally, we illustrate detailed ROC curves for three representative datasets: CARDIO, WAVEFORM, and KDDCUP99, in Figure 1. Note that the subspace hashing solutions, OS-Hash and RS-Hash, achieve consistently better

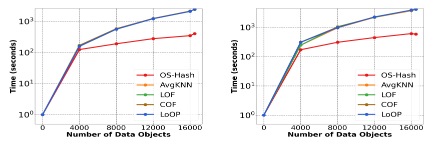
Table 4: AUC results for time series datasets.

Datasets	OS-Hash	AvgKNN	LOF	COF	LoOP
PICKUP	**75.00**	74.50	73.00	68.50	63.75
PEBBLE	**79.17**	7.11	41.02	49.59	51.14
POWER	**66.03**	52.10	37.43	40.82	38.89
ECG5000	**92.17**	74.41	72.14	69.26	69.35
CROP	**83.96**	66.82	35.68	38.32	36.78



(a) PEBBLE (b) ECG5000 (c) CROP

Figure 3: ROC curves in time series datasets.



(a) CROP (b) UNIFORM(0,1)

Figure 4: Runtime costs in time series datasets.

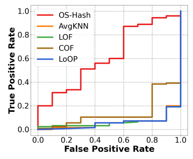
accuracy results than other outlier detection methods, and the performance gains can be quite notable in all datasets. In addition, OS-Hash achieves better AUC results in three datasets: CARDIO, MUSK, and WAVEFORM, than RS-Hash; it outperforms RS-Hash with 25.02% improvement in accuracy in the WAVEFORM dataset, These results advocate that oblique subspaces create a new perspective to identify intrinsic outliers from multidimensional data with well-defined schemas or attributes, and OS-Hash can be a feasible, and sometimes better, solution for subspace outlier detection in high-dimensional datasets.

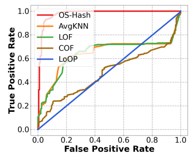
We then examine the runtime costs of different outlier detection methods on the largest dataset, KDDCUP99. We sample data objects at random in order to create a series of datasets of varying sizes. In addition, we construct two more synthetic datasets from KDDCUP99 with varying values and distributions for each dimension: NORMAL$(0, 1)$ contains real values drawn from the standard normal distribution; UNIFORM$(0, 1)$ contains real values following the uniform distribution in the range of $[0, 1]$. Both NORMAL$(0, 1)$ and UNIFORM$(0, 1)$ have the same number of data objects and dimensions as KDDCUP99. The runtime costs for these datasets are reported in Figure 2. Remark that OS-Hash and RS-Hash are the most efficient methods; both can accomplish outlier detection within 10 seconds on all datasets, and are orders-of-magnitude faster than the remaining methods (Note the y-axis represents runtime costs in log-scale). In addition, OS-Hash demonstrates linear scalability across different datasets.

We then focus on arbitrary-shaped data without pre-specified schemas or well-defined dimensions, such as time series and graphs. In this setting, the state-of-the-art solution, RS-Hash, simply fails. We first examine the AUC results of five different outlier detection methods on five time-series datasets, as reported in Table 4. In addition, we present detailed ROC curves of these methods on three representative datasets: PEBBLE, ECG5000, and CROP, and the results are illustrated in Figure 3. It is clear that OS-Hash consistently

Table 5: AUC results for graph datasets.

Datasets	OS-Hash	AvgKNN	LOF	COF	LoOP
MUTAG	**61.62**	5.76	6.04	13.84	5.52
FINGER	**55.03**	51.59	41.38	48.43	51.59
AIDS	**97.51**	64.43	64.22	48.09	49.75
MUTAGEN	**63.52**	56.51	55.14	60.40	58.05
TOX21	**71.97**	49.58	49.67	50.51	50.00


(a) MUTAG


(b) AIDS

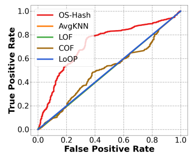
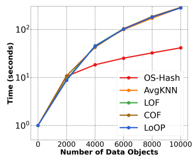
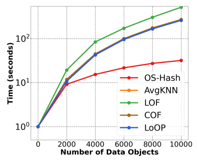

(c) TOX21

Figure 5: ROC curves in graph datasets.


(a) TOX21


(b) EDGE18

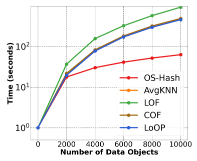

(c) NODE18

Figure 6: Runtime results in graph datasets.

achieves the highest accuracy results for subspace outlier detection across all time-series datasets, and the performance gains are notable when compared with the other four baseline methods. For instance, in the ECG5000 dataset, the AUC of OS-Hash is 23.87% higher than that of the best baseline, AvgKNN.

We further report runtime costs of different methods on the largest CROP dataset. Additionally, we create another synthetic dataset, denoted UNIFORM$[0, 1]$, from CROP by generating time-varied real values following the uniform distribution in the range of $[0, 1]$. The runtime results on both datasets are illustrated in Figure 4. Note that OS-Hash exhibits linear-time scalability, and is one order-of-magnitude faster than the other four baseline methods. This performance gap is more notable when the datasets get larger, demonstrating the clear advantages of OS-Hash for outlier detection in large time-series datasets.

We then consider subspace outlier detection on graph data. Table 5 presents the AUC results of five outlier detection methods on five graph datasets, and Figure 5 further illustrates the ROC curves of these methods on three representative datasets: MUTAG, AIDS, and TOX21. For graph data, OS-Hash consistently achieves the best accuracy results in subspace outlier detection, and the performance gains are significant compared with the other four baseline methods. In the AIDS dataset, for instance, OS-Hash obtains 51.34% improvement in outlier detection accuracy compared to the best baseline, AvgKNN.

We further evaluate the runtime costs of different methods on the largest graph dataset, TOX21. Additionally, we create two new synthetic datasets from TOX21, whose average numbers of nodes and edges are both 18: one dataset, EDGE18, contains $10,000$ graphs, each of which has a varying number of nodes and a fixed number of 18 edges; the other, NODE18, contains $10,000$ graphs, each of which

Table 6: AUC results in multidimensional data streams.

Datasets	OS-Stream	RS-Stream	AvgKNN-Stream	LOF-Stream
ACTIVITY	**99.96**	84.51	33.59	38.55
KDDCUP99-T	87.09	**95.27**	12.43	66.35

Table 7: AUC results in time-series streams.

Dataset	OS-Stream	AvgKNN-Stream	LOF-Stream
ACTIVITY-T	**81.89**	35.08	40.57
CROP-T	**81.99**	71.89	52.97

Table 8: AUC results in graph streams.

Datasets	OS-Stream	AvgKNN-Stream	LOF-Stream
TOX21-AR-T	**72.07**	52.87	53.64
MCF-7-T	**60.08**	52.94	56.18

has 18 nodes and a varying number of edges. The runtime costs on these three datasets are illustrated in Figure 6. Again, our linear-time solution, OS-Hash, achieves an order-of-magnitude speedup for subspace outlier detection in graphs compared with the other four baseline methods.

5.5 Results in The Data Stream Setting

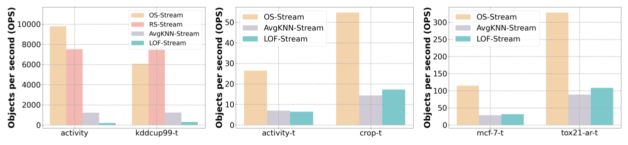
We then report our experimental results in the data stream setting, where data objects arrive at high speed and outliers need to be identified in real time without repeated access to the underlying data streams. To this end, very few outlier detection methods that work in the static setting can be extended to data streams, where objects of the data stream are either multidimensional or arbitrary-shaped. Specifically, RS-Stream (the streaming variant of RS-Hash) can only work for the case of multidimensional data streams. In contrast, our solution, OS-Stream, is generic enough and applicable for all different types of data streams.

We first present the outlier detection accuracy results on two multidimensional data streams, and the AUC results are shown in Table 6. Note that our solution, OS-Stream, outperforms other distance- or density-based outlier detection methods with 1.31× to 7× improvements in terms of outlier detection accuracy, and its accuracy results are comparable to those of RS-Stream. However, for arbitrary-shaped data streams, such as time-series data streams (in Table 7), and graph streams (in Table 8), we recognize that OS-Stream achieves the best outlier detection performance, which is significantly better than other baseline methods for real-time outlier detection in data streams.

We further evaluate the efficiency of different outlier detection methods across three types of data streams, and the results are shown in Figure 7. Remark that in these experiments, we consider the throughput, the number of data objects processed per second, as the main efficiency metric. OS-Stream achieves the highest throughput across all different data streams. Specifically, in the time-series and graph streams, the efficiency gain of OS-Stream is at least 3.56× when compared with AvgKNN-Stream and LOF-Stream. As a result, OS-Stream is a highly efficient method that can be deployed in real-world data streams of any data type for real-time subspace outlier detection.

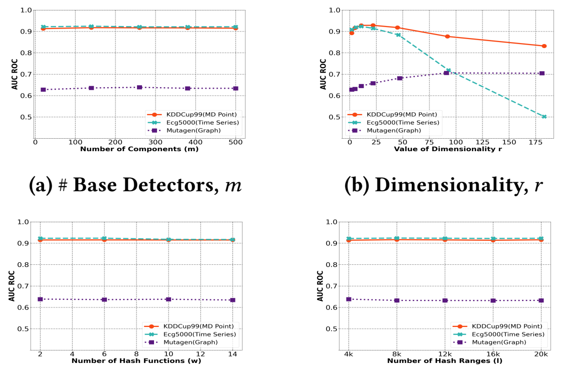
5.6 Parameter Analysis

There exist several algorithmic parameters for OS-Hash, including (1) the number of base outlier detectors, m, or equivalently, the



(a) Multidimensional **(b) Time Series** **(c) Graphs**

Figure 7: Runtime results in data streams.



(a) # Base Detectors, m **(b) Dimensionality, r**

(c) # Hash Functions, w **(d) Hash Table Size, l**

Figure 8: AUC results vs. parameters of OS-Hash.

number of components in the ensemble of OS-Hash; (2) the dimensionality r of the oblique subspaces, or equivalently, the number of pairs of data objects sampled from O that identify the oblique subspace for each base detector; (3) the number of pairwise independent hash functions, w, in the CountMin sketch; (4) the number of hash table entries, l, or equivalently, the hash table size of the CountMin sketch. In this section, we explore how these parameters are regulated in order to obtain good outlier detection performance for OS-Hash. We select one multidimensional dataset, KDDCUP99, one time series dataset, ECG5000, and one graph dataset, MUTAGEN, in the experiments, and outlier detection accuracy results *w.r.t.* different algorithm parameters are illustrated in Figure 8.

First of all, as presented in Figure 8(a), by tuning the number of base detectors, m, from 20 to 500, we witness very marginal changes in the outlier detection accuracy. Meanwhile, the runtime costs grow proportionally because more base detectors are involved in our ensemble solution. In practice, $m = 100$ is sufficient across different datasets of varying data types. We then examine the effects of the oblique subspace dimensionality, r, shown in Figure 8(b). By increasing the values of r from 5 up to 180, we recognize an improvement for the outlier detection accuracy at first, followed by a steady decline. The primary reason is that, the localized outliers can be easily represented and identified within a sufficient number ($20 \leq r \leq 50$) of oblique subspace dimensions. However, when the dimensionality keeps increasing, we are facing with the so-called “curse-of-dimensionality" problem: very few data objects will be mapped to the same CountMin sketch entry across all r oblique dimensions, thus leading to poor outlier detection performance. We further explore the two parameters, the number of pairwise independent hash functions, w, and the hash table size, l, which determine accuracy and space cost of the CountMin sketch, and the experimental results are presented in Figure 8(c) and Figure 8(d), respectively. Note that the outlier detection accuracy remains stable given different values of w and l across all datasets. In practice, we set $w = 4$ and $l = 10,000$ in order to strike the right balance between accuracy and space costs of the CountMin sketch.

6 Related Work

Outlier detection has been a fundamental data mining task [2] with a series of classic solutions including statistical methods [8], distance-based methods [6, 7, 9, 30], density-based methods [11, 29], pattern compression methods [5], and spectral methods [13, 31]. Some of them have been generalized to the streaming environment, including distance-based methods [6] and tree-based methods [37, 42]. Due in particular to the curse of dimensionality and the existence of irrelevant dimensions, traditional methods suffered from performance degradation and computational inefficiency when working on high-dimensional data. To address these limitations, subspace outlier detection methods were devised to identify outliers from within a subset of *relevant* dimensions [4], or in low-dimensional manifolds embedded in high-dimensional data [31].

One challenge in subspace outlier detection is that a single subspace is often too weak to account for all outliers from the datasets. The ensemble solutions [3, 21] addressed this limitation by discovering outliers through multiple views of data. Some studies carried out subspace outlier detection in multiple axis-parallel subspaces [17, 21]; Others used statistical methods to identify relevant subspaces [26, 27]. However, existing solutions primarily combined feature bagging with distance-based methods, which are not well-suited for integration. Our method, OS-Hash, proposes a bi-sampling approach that combines both data sampling and dimension sampling, making it better suited for a perturbed oblique subspace hashing methodology as discussed in this paper.

The state-of-the-art subspace hashing solution, RS-Hash [32], identified outliers in axis-parallel subspaces, confining it to the applications of multidimensional data only. In contrast, our work, OS-Hash, extends axis-parallel subspaces to oblique subspaces. As a result, OS-Hash is applicable to not only multidimensional data but also time series, graphs, and any arbitrary-shaped data types in both static and streaming settings, as long as the pairwise data similarity/distance functions are properly defined. To the best of our knowledge, OS-Hash is the first to enable outlier detection in oblique subspaces.

7 Conclusion

Subspace outlier detection has been a fundamental data mining task that identifies outliers from high-dimensional datasets. State-of-the-art solutions, however, can only detect outliers from axis-parallel subspaces for multidimensional data with well-defined attributes or dimensions. In this work, we introduced a new notion of oblique subspaces that enabled outlier detection for arbitrary-shaped data as long as the pairwise proximity/distance can be defined between data objects. Our outlier detection solution, OS-Hash, is a linear-time, constant-space method general enough for efficient and cost-effective subspace outlier detection on both multidimensional and arbitrary-shape data, such as time-series and graphs, and it can be extended to the case of data streams. Our experimental studies in real-world datasets and data streams validated the generality, accuracy, and efficiency of OS-Hash, which outperformed state-of-the-art solutions for subspace outlier detection.

8 GenAI Usage Disclosure

There was no use of Generative AI tools in any stage of the proposed research, including the code, data, and the writing of this paper.

References

[1] Charu C. Aggarwal. 2006. *Data Streams: Models and Algorithms*. Springer Inc., Berlin, Heidelberg.

[2] Charu C. Aggarwal. 2016. *Outlier Analysis* (2nd ed.). Springer Inc.

[3] Charu C. Aggarwal and Saket Sathe. 2015. Theoretical Foundations and Algorithms for Outlier Ensembles. *SIGKDD Explor. Newsl.* 17, 1 (2015), 24–47.

[4] Charu C Aggarwal and Philip S Yu. 2001. Outlier detection for high dimensional data. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD'01)*. 37–46.

[5] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. 2012. Fast and reliable anomaly detection in categorical data. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM'12)*. 415–424.

[6] Fabrizio Angiulli and Fabio Fassetti. 2007. Detecting distance-based outliers in streams of data. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM'07)*. 811–820.

[7] Fabrizio Angiulli and Clara Pizzuti. 2002. Fast Outlier Detection in High Dimensional Spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'02)*. 15–26.

[8] Vic Barnett and Toby Lewis. 1994. *Outliers in Statistical Data*. John Wiley & Sons.

[9] Stephen D Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'03)*. 29–38.

[10] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier Detection: Methods, Models, and Classification. *ACM Comput. Surv.* 53, 3 (2020), 1–37.

[11] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. *SIGMOD Rec.* 29, 2 (2000), 93–104.

[12] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75.

[13] Xuan Hong Dang, Barbora Micenková, Ira Assent, and Raymond T Ng. 2013. Outlier detection with space transformation and spectral analysis. In *Proceedings of the 2013 SIAM International Conference on Data Mining (SDM'13)*. 225–233.

[14] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2018. *Data Stream Management: Processing High-Speed Data Streams*. Springer Inc.

[15] Omer Gold and Micha Sharir. 2018. Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier. *ACM Trans. Algorithms* 14, 4 (2018), 1–17.

[16] Zhao Kang, Yiwei Lu, Yuanzhang Su, Changsheng Li, and Zenglin Xu. 2019. Similarity learning via kernel preserving embedding. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI'19)*. 4057–4064.

[17] Fabian Keller, Emmanuel Muller, and Klemens Bohm. 2012. HiCS: High Contrast Subspaces for Density-Based Outlier Ranking. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE'12)*. 1037–1048.

[18] Eamonn J. Keogh and Michael J. Pazzani. 2000. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*. 285–289.

[19] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2009. LoOP: local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM'09)*. 1649–1652.

[20] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. 2008. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. 444–452.

[21] Aleksandar Lazarevic and Vipin Kumar. 2005. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD'05)*. 157–166.

[22] John A. Lee and Michel Verleysen. 2009. Simbed: Similarity-Based Embedding. In *Proceedings of the 19th International Conference on Artificial Neural Networks (ICANN'09)*. 95–104.

[23] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* 6, 1 (2012), 1–39.

[24] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. 2018. xStream: Outlier Detection in Feature-Evolving Data Streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. 1963–1972.

[25] Abdullah Mueen and Eamonn Keogh. 2016. Extracting Optimal Performance from Dynamic Time Warping. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 2129–2130.

[26] Emmanuel Müller, Ira Assent, Patricia Iglesias, Yvonne Mülle, and Klemens Böhm. 2012. Outlier ranking via subspace analysis in multiple views of the data. In *2012 IEEE 12th international conference on data mining (ICDM'12)*. 529–538.

[27] Emmanuel Müller, Matthias Schiffer, and Thomas Seidl. 2011. Statistical selection of relevant subspace projections for outlier ranking. In *2011 IEEE 27th international conference on data engineering (ICDE'11)*. 434–445.

[28] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. 2022. Graph Kernels: A Survey. *J. Artif. Int. Res.* 72 (2022), 943–1027.

[29] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. 2003. Loci: Fast outlier detection using the local correlation integral. In *Proceedings 19th international conference on data engineering (ICDE'03)*. 315–326.

[30] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD'00)*. 427–438.

[31] Saket Sathe and Charu Aggarwal. 2016. LODES: Local density meets spectral outlier detection. In *Proceedings of the 2016 SIAM international conference on data mining (SDM'16)*. 171–179.

[32] Saket Sathe and Charu C. Aggarwal. 2018. Subspace histograms for outlier detection in linear time. *Knowl. Inf. Syst.* 56, 3 (2018), 691–715.

[33] Bernhard Scholkopf and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

[34] Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel. 2022. A generalized Weisfeiler-Lehman graph kernel. *Machine Learning* 111, 7 (2022), 2601–2629.

[35] John Shawe-Taylor and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

[36] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.* 12 (2011), 2539–2561.

[37] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast anomaly detection for streaming data. In *Twenty-second international joint conference on artificial intelligence (IJCAI'11)*. 1511–1516.

[38] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David Wai-Lok Cheung. 2002. Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD'02)*. 535–548.

[39] Daniel Ting. 2018. Count-Min: Optimal Estimation and Tight Error Bounds using Empirical Error Distributions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. 2319–2328.

[40] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. 2019. *Wasserstein weisfeiler-lehman graph kernels*. 6439–6449.

[41] Luan Tran, Min Y. Mun, and Cyrus Shahabi. 2020. Real-time distance-based outlier detection in data streams. *Proc. VLDB Endow.* 14, 2 (2020), 141–153.

[42] Ke Wu, Kun Zhang, Wei Fan, Andrea Edwards, and S Yu Philip. 2009. Rs-forest: A rapid density estimator for streaming anomaly detection. In *2014 IEEE International Conference on Data Mining (ICDM'09)*. 600–609.

[43] Arthur Zimek, Ricardo J.G.B. Campello, and Jörg Sander. 2014. Ensembles for unsupervised outlier detection: challenges and research questions. *SIGKDD Explor. Newsl.* 15, 1 (2014), 11–22.