Knowledge and Information Systems (2023) 65:2991–3022
https://doi.org/10.1007/s10115-023-01854-1

REGULAR PAPER

BPF: a novel cluster boundary points detection method for static and streaming data

Vijdan Khalique[1] · Hiroyuki Kitagawa[2,3] · Toshiyuki Amagasa[4]

Received: 3 October 2022 / Revised: 19 December 2022 / Accepted: 27 February 2023 /
Published online: 21 March 2023

Abstract
Data points situated near a cluster boundary are called *boundary points* and they can represent useful information about the process generating this data. The existing methods of boundary points detection cannot differentiate boundary points from outliers as they are affected by the presence of outliers as well as by the size and density of clusters in the dataset. Also, they require tuning of one or more parameters and prior knowledge of the number of outliers in the dataset for tuning. In this research, a boundary points detection method called *BPF* is proposed which can effectively differentiate boundary points from outliers and core points. *BPF* combines the well-known outlier detection method *Local Outlier Factor* (LOF) with *Gravity* value to calculate the BPF score. Our proposed algorithm *StaticBPF* can detect the top-m boundary points in the given dataset. Importantly, *StaticBPF* requires tuning of only one parameter i.e. the number of nearest neighbors (k) and can employ the same k used by LOF for outlier detection. This paper also extends *BPF* for streaming data and proposes *StreamBPF*. *StreamBPF* employs a grid structure for improving k-nearest neighbor computation and an incremental method of calculating BPF scores of a subset of data points in a sliding window over data streams. In evaluation, the accuracy of *StaticBPF* and the runtime efficiency of *StreamBPF* are evaluated on synthetic and real data where they generally performed better than their competitors.

✉ Hiroyuki Kitagawa
kitagawa@cs.tsukuba.ac.jp

Vijdan Khalique
khalique.vijdan@kde.cs.tsukuba.ac.jp

Toshiyuki Amagasa
amagasa@cs.tsukuba.ac.jp

[1] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan

[2] International Institute for Integrative Sleep Medicine, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan

[3] National Institute of Advanced Industrial Science and Technology, 2-3-26 Aomi, Koto-ku, Tokyo 135-0064, Japan

[4] Center for Computational Sciences, University of Tsukuba, 1-1-1, Tennodai, Tsukuba, Ibaraki 305-8573, Japan

Keywords Boundary points detection · Streaming data · Data mining · Cluster boundary

1 Introduction

Clustering is one of the data mining techniques that divides a dataset into subsets such that data belonging to each subset have some similar properties [24]. These clusters of data may represent a useful phenomenon. In cluster analysis useful features of data is extracted. For example, in a customer dataset, a cluster of data objects may represent a specific behavior of customers, or in an image dataset, a cluster of images may share similar properties.

Outlier detection is related to clustering as an outlier is defined as a data object that does not belong to any cluster and deviates from the majority of the data objects [15]. An outlier is often situated in an isolated region of the data space. Many techniques have been proposed for outlier detection based on different properties of the dataset. For example, distance [3, 20], density [5, 17], angle [23, 32] and isolation [14, 28].

There are several research efforts targeting the problem of clustering [1, 7, 12] and outlier detection [5, 29, 33, 39] in static and streaming data. However, a limited research has been dedicated to the *boundary points detection*. In [26], border or boundary points are defined as points which are located at the extremes of a class region or near free pattern space. In other words, boundary points are located at the border of a cluster forming its boundary. Hence, boundary points detection can be defined as the task of detecting the points which are situated at the boundary of a cluster [41].

Detecting boundary points may provide useful information about the system which is generating this data. Consider the example of a disease detection system in which the normal data objects may represent healthy patients and the patient who have contracted a certain disease may be represented by outliers. In this example, the boundary points may represent normal patients showing abnormal symptoms but somehow have not yet developed the disease. Consequently, closely monitoring such boundary cases may reveal interesting information about the disease. Similar motivating examples have been presented in the related papers [9, 27, 34, 41].

Amongst many outlier detection techniques [21, 23, 28, 30], Local Outlier Factor (LOF) [5] is one of the most popular and competitive density-based outlier detection methods [43]. It can detect outliers based on relative density of a target object according to its local neighborhood. Core points are situated in the inner region of the cluster, whereas the outliers are isolated points in the less dense regions. LOF can detect outliers by calculating the LOF scores where outliers get larger LOF scores (> 1) than other points.

The problem of boundary points detection requires the detection of boundary points while ignoring the core points and outliers. In our previous work [19] , we proposed *Boundary Point Factor (BPF)* method which calculates a boundary point score called BPF score to effectively identify boundary points. In a nutshell, *BPF* calculates the BPF score of a given point by taking the ratio of its *Gravity* value and *LOF* score. The Gravity value is calculated by the norm of the average of unit vectors from the given point to its k-nearest neighbors. Based on our proposed formulation, BPF scores of boundary points tend to be greater than outliers and core points. As a result, the boundary points can be distinguished from the outliers and core points based on the BPF scores. We propose *BPF* algorithm for static datasets which calculates BPF scores and output the top-m boundary points. Followings are the advantages of our proposed method [19]:

- *BPF* is robust to the presence of outliers and clusters of different sizes and densities.

- *BPF* can be used with LOF for detecting boundary points and outliers as *BPF* shares the k-nearest neighbors and LOF computation.
- *BPF* has one tune-able parameter k (number of nearest neighbors) where the value of k tuned for boundary points detection with BPF can also be used for outlier detection with LOF.

This paper extends *BPF* to the problem of boundary points detection over data streams and provides more intensive experimental results to verify the effectiveness of *BPF*. It is important to clarify that in our previous work, *BPF* combinedly represented the method of calculating the BPF score, and the algorithm that output the top-m boundary points in a static dataset. In this work, we refer to the algorithm of detecting the top-m boundary points via *BPF* as *StaticBPF*, whereas *BPF* refers to the method of calculating BPF score by combining Gravity and LOF.

In streaming data, the task of boundary points detection becomes more challenging due to high arrival rate of data. As a result, a faster method is desirable which can calculate the BPF scores of new arrival points and update the BPF scores of points affected by arrival of new points or expiration of old points. This paper proposes a grid-based runtime efficient method to address the problem of fast boundary points detection over data streams. The challenges are to improve the computation of the k-nearest neighbors and BPF scores of the new arrival points and the points affected by arrival and expiration of points due to window slide. To address this problem, we propose *StreamBPF* which employs a grid structure to efficiently compute the k-nearest neighbors and uses an incremental method of computing BPF scores. This paper is an extension of our previous work [19], and followings are the key extensions in this work:

- Quantitative evaluation of *StaticBPF* on 2- and high-dimensional synthetic and real data. In our previous work, we demonstrated the accuracy by showing the detected boundary points on 2-d synthetic data and real data. In this paper, in addition to the previous results, the accuracy results are shown quantitatively w.r.t. precision, recall, F1 score, area under precision-recall curve (AUC PR) and area under ROC curve (AUC ROC) on all datasets.
- Proposal of a boundary points detection method over data stream named *StreamBPF*. Our previous contribution is suitable for static datasets, and it is computationally expensive to use it for streaming data. Therefore, in this paper, we propose $StreamBPF$ that:
  1. uses a grid structure to improve the k-nearest neighbors computation, and
  2. incrementally computes BPF scores adopting observations in [33].
- Runtime performance evaluation of *StreamBPF* on synthetic and real data, and comparison with *StaticBPF* and other methods.

2 Related work

BORDER [41] is one of the boundary points detection algorithms. It exploits the observation that boundary points have smaller number of reverse k-nearest neighbors (RN_k) than core points and identifies boundary points. However, the computation of RN_k is expensive, and therefore they proposed to use G-ordering kNN join method [40] to improve the computation of RN_k. BORDER was found to be effective in datasets which do not have outliers. In the case of dataset with outliers, BORDER cannot differentiate between outliers and boundary points, as both of them tend to have a smaller number of RN_k. To address the shortcoming of BORDER, BRIM [35] proposed to consider *eps*-neighborhood to successfully detect

the boundary points in datasets with many outliers. Given a distance *eps*, BRIM uses the observation that since a boundary point is located at the edge of a dense region, its *eps*-neighborhood can be distributed in either positive or negative direction based on the diameter line which divides its *eps*-neighborhood into two parts. Furthermore, boundary points tend to have denser *eps*-neighborhood than outliers. The major drawback of BRIM is that it cannot perform well in datasets with clusters of different densities and scales due to the fixed *eps* value.

Recently, Li et al. proposed BPDAD [27] for detecting outliers and boundary points based on geometrical measures. BPDAD exploits two important observations that outliers and boundary points have lower local densities and smaller variance of angles than their neighbors. Consequently, BPDAD output outliers and boundary points together and it does not specifically detect boundary points. BorderShift [9] is another boundary points detection algorithm which uses similar observations regarding the densities of outliers, boundary and core points. BorderShift employs Parzen Window (kernel density estimation) to estimate the local density of a point and MeanShift vector to determine the direction of dense region. It effectively detects boundary points provided that its three parameters (k, λ_1 and λ_2) are tuned appropriately. Particularly, tuning λ_1 and λ_2 can be difficult as it requires prior information about the number of outliers in the dataset.

For high-dimensional data, [6, 34] project high-dimensional data onto lower dimensions for boundary points detection. However, similar to BorderShift [9], their parameter tuning depends on the prior knowledge of the number of outliers in datasets. A more desirable technique is easy to tune and does not require any prior information about the data distribution of the given dataset. It should take a dataset with or without outliers as input and output the top-m boundary points.

Our previous work [19] introduced *BPF* method and experimentally showed its effectiveness for static datasets. However, in [19], we did not consider the problem to detect boundary points occurring in streaming data. To the best of our knowledge, there is no proposed method of boundary points detection for streaming data. Since, *LOF* is one of the key components of *BPF*, incremental computation of *LOF* can improve the runtime performance of *BPF* and may make it suitable for streaming data.

There are many outlier detection methods based on different observations for streaming data [2, 8, 16, 18, 22, 39, 42]. ILOF is the extension of LOF for data streams which can incrementally update the LOF scores of the points [33]. ILOF proposed two algorithms: Insertion and Deletion which are used when a new point is inserted and deleted from the dataset, respectively. One disadvantage of ILOF is that it requires a large amount of memory. Consequently, to improve the space complexity, [37] proposed a memory efficient method called MiLOF which stores the summary of past data to improve memory consumption. Since MiLOF stores the summary of past data as cluster centers, its accuracy may degrade with time. DILOF [29] addressed this problem by preserving the density information of past data. Another attempt to improve LOF for streaming data is [13] which employs a cube-based method to approximate the LOF scores of incoming points. All these proposed methods are approximation of ILOF algorithm. ILOF is related to our proposed method for streaming data as we need to update LOF scores in order to update the BPF scores of points. Hence, we adopt the observations given in [33] to update the LOF scores. Furthermore, we propose a grid structure to improve the runtime of the k-nearest neighbors computation.

Table 1 List of important symbols

Symbols	Description
k	# Nearest neighbors
n	# Points in dataset or window
m	# Boundary points
d	Dimensionality of data
D	Dataset of d-dimensional data points
W_t	Window at time step t
w	Slide size of a count-based window
$N_k(p)$	Set of k-nearest neighbors of point p
$RN_k(p)$	Set of reverse k-neighbors of point p
$kdist(p)$	Distance of p with its kth nearest neighbor
$dist(p, q)$	Distance between point p and q
$G(p)$	Gravity value of point p
$LOF(p)$	LOF score of point p
l	Grid cell length
$\mathbb{G}_t$	Grid at time step t
C_i	$C_i = (K_i, \mathbb{S}_i)$ represents a cell in $\mathbb{G}_t$
K_i	Cell key of cell C_i
$\mathbb{S}_i$	Subset of points in W_t belonging to cell C_i

3 Preliminaries

This section briefly introduces the definitions related to Local Outlier Factor (LOF). The readers may refer to [5] for further details. Furthermore, (Table 1) shows the list of important symbols used in this paper.

Given a d-dimensional point p in the dataset D, let k represent the number of nearest neighbors. The LOF score of p ($LOF(p)$) can be calculated using two key concepts: Reachability Distance $reach\text{-}dist_k(p, o)$ and Local Reachability Density $lrd_k(p)$. The following definitions present these concepts followed by the definition of LOF.

Definition 1 (*Reachability distance*) Reachability Distance of a point p w.r.t. point o is defined as:

$$reach\text{-}dist_k(p, o) = \max\{kdist(o), dist(p, o)\}, \tag{1}$$

where $kdist(o)$ is the distance from o to its kth neighbor and $dist(p, o)$ is the distance between point p and o.

Definition 2 (*Local reachability density*) Local Reachability Density of a point p denoted as $lrd_k(p)$ is defined as:

$$lrd_k(p) = 1/\left(\frac{\sum_{o \in N_k(p)} reach\text{-}dist_k(p, o)}{|N_k(p)|}\right), \tag{2}$$

where $N_k(p)$ is the set of k-nearest neighbors of p and $|N_k(p)|$ represents the cardinality of $N_k(p)$.

Intuitively, the local reachability density is the estimation of density of p w.r.t. its neighbors $o \in N_k(p)$. More concretely, $lrd_k(p)$ is the reciprocal of average reachability distance from p

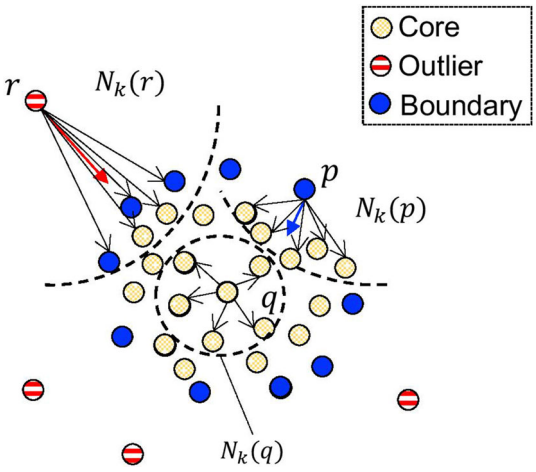


Fig. 1 Core and boundary points form the dense region, and outliers are isolated (color figure online)

to its k-nearest neighbors. Therefore, larger the reachability distances of p, smaller is $lrd_k(p)$. Based on Definitions 1 and 2, we can define the local outlier factor (LOF).

Definition 3 (*Local Outlier Factor (LOF)*) Local Outlier Factor of a point p is defined as:

$$LOF(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|}. \tag{3}$$

$LOF(p)$ is the outlier factor of the point p which indicates its degree of outlierness. If p is a core point then $LOF(p)$ is close to 1, and if p is a boundary point, then $LOF(p)$ is slightly greater than core points but still close to 1. In case p is an outlier, $LOF(p)$ is greater than 1. The details about the range of LOF score are explained in [5].

4 Boundary point factor (*BPF*)

This section introduces the definitions related to the Boundary Point Factor (*BPF*) method and explains the basic observations about it. Firstly, the definition of Gravity $G(p)$ of a point is given.

Definition 4 (*Gravity*) Given a point $p \in D$, the set of k-nearest neighbors of p $N_k(p)$, and the norm $\| \cdot \|$, Gravity of p can be defined as:

$$G(p) = \frac{1}{|N_k(p)|} \left\| \sum_{o \in N_k(p)} \frac{\overrightarrow{po}}{\|\overrightarrow{po}\|} \right\|. \tag{4}$$

Intuitively, Gravity ($G(p)$) is a scalar value which indicates how the neighborhood of p is distributed. Consider the boundary point p shown in Fig. 1. Taking the average of unit vectors originating from p to its k-nearest neighbors will result in a single vector (blue arrow). Then, calculating $G(p)$ will result in a larger Gravity value than the core point q as it is surrounded by points in all directions which will result in a smaller $G(q)$. Generally, the Gravity value of an outlier depends on the data distribution. Hence, the following inequality is expected to

hold for boundary and core points:

$$G(q) < G(p). \tag{5}$$

Similarly, the LOF scores of data points w.r.t. their k-neighborhood can be calculated using Definition 3. As shown in [5], LOF scores of core and boundary points are close to 1 ($LOF(q), LOF(p) \approx 1$) and LOF scores of outliers are greater than 1. Consequently, for the points p, q and r shown in Fig. 1, the following inequality is expected to hold:

$$LOF(q), LOF(p) < LOF(r). \tag{6}$$

Based on these observations, Boundary Point Factor (BPF) score is defined as follows.

Definition 5 (*Boundary Points Factor (BPF) score*) Given a point $p \in D$, $LOF(p)$ and Gravity $G(p)$, Boundary Point Factor score of p $BPF(p)$ can be calculated as follows:

$$BPF(p) = \frac{G(p)}{LOF(p)}. \tag{7}$$

From Definition 5, the following inequality is expected to hold:

$$BPF(q), BPF(r) < BPF(p). \tag{8}$$

Namely, BPF scores of boundary points are more likely to be greater than core points and outliers. Hence, boundary points in a dataset can be identified using BPF scores.

5 *StaticBPF*

BPF can be applied on static datasets for boundary points detection. The algorithm that uses *BPF* for detecting the top-m boundary points from static datasets is named as *StaticBPF*. This section presents the algorithm steps and runtime complexity of *StaticBPF*, and shows the results of accuracy evaluation.

5.1 Algorithm

The main idea of *StaticBPF* algorithm is to calculate BPF scores of all points in the dataset D. Firstly, the algorithm calculates k-neighborhood of all points in D. Next, for each point in D, it calculates BPF score based on LOF and Gravity according to Definition 5. After calculating BPF scores, the algorithm sorts all points in the descending order of the BPF scores. Given the parameter m, *StaticBPF* will output the list $\mathbb{C}$ of the top-m boundary points in D. The steps are given in Algorithm 1.

5.2 Runtime complexity

Let n represent the number of points in a d-dimensional dataset. The most computationally expensive task for *StaticBPF* is the k-nearest neighbors search for each point which have the complexity of $O(n^2 d)$. However, the runtime complexity can be improved to $O(n \log n)$ by using a suitable indexing technique [31, 36]. This runtime complexity can be applied to other boundary points detection methods like BORDER [41], BRIM [35], BPDAD [27] and BorderShift [9] as well. As explained, BPF score is calculated based on the Gravity and LOF scores of n points, where the complexity of calculating Gravity and LOF scores can be given

Algorithm 1 *StaticBPF*

Require: Dataset D, #nearest neighbors k, #boundary points m
1: **for** $p \in D$ **do**
2: $N_k(p) \leftarrow k$-nearest neighbors of p.
3: **end for**
4: **for** $p \in D$ **do**
5: calculate $BPF(p)$ using Definition 5 and store in $\mathbb{B}$.
6: **end for**
7: Sort $\mathbb{B}$ in descending order of BPF scores.
8: $\mathbb{C} \leftarrow$ top-m points in $\mathbb{B}$. /* list of top-m boundary points in D*/
return $\mathbb{C}$

as $O(nkd)$ and $O(nk)$, respectively. Hence, the overall runtime complexity of *StaticBPF* algorithm without using indexing structure is $O(n^2d + nkd + nk)$.

5.3 Evaluation of *StaticBPF*

In this section, we show experimental evaluation of *StaticBPF*. The experiments are conducted on 2- and high-dimensional synthetic datasets as well as on real datasets.

5.3.1 Experimental setup

This section explains the steps of obtaining the ground truth and tuning the parameters for all methods.

In order to perform quantitative evaluation, it is fundamental to have the ground truth boundary points for all datasets. Therefore, we applied the following steps on each synthetic and real dataset to obtain the ground truth:

1. Apply DBSCAN [12] to identify clusters/classes and outliers in the dataset. Remove the outliers from the dataset.
2. Apply BORDER [41] on each identified cluster/class in the dataset. Consider the points as boundary points which have the boundary scores less than or equal to $\alpha\%$ of the average boundary score of that cluster. Repeat this process for each cluster in the given dataset.
3. Consider all the points obtained in step 2 as the top m ground truth boundary points of the dataset.

It may be noted that the outliers detected by DBSCAN are removed temporarily to apply BORDER and detect the top-m boundary points. After that, the removed outliers are inserted again in the dataset.

DBSCAN is used to detect the outliers from all datasets. In synthetic datasets, we randomly introduced a fixed number of outliers. However, there may exist outliers relative to the clusters. Therefore, we applied DBSCAN to obtain all outliers in the given dataset. The parameters *MinPts* and *eps* of DBSCAN are tuned using the *k-distance* plot as suggested in [38]. We checked the "elbow" values of *eps* at fixed *MinDist* and chose the appropriate *eps* such that the number of outliers detected by DBSCAN are greater or equal to the number of randomly introduced outliers. In real datasets, we chose a large value of *eps* to detect the outstanding outliers.

In order to obtain the ground truth boundary points via BORDER, we need to choose an appropriate value of its parameter k. To do so, we tuned k of BORDER for detecting the outliers according to the ground truth outliers obtained by DBSCAN in step 1. The value of k which gives the best accuracy in terms of precision, recall and F1 is used to obtain the

Table 2 Range of parameter values used for tuning all methods

Method	Range
StaticBPF	$k = [50, 100, 150, 200, 250]$
BorderShift	$k = [5, 10, 50, 100]$
BRIM	*eps* = [*min_distance*, *avg_distance*]
BORDER	$k = [50, 100, 150, 200, 250]$

ground truth boundary points in step 2. It is reasonable as BORDER uses the observation that boundary points have a smaller number of reverse k-neighbors than core points. However, this observation is also true for outliers. Hence, tuning k of BORDER for outlier detection on a dataset and then applying BORDER with the tuned k on the same dataset after removing the outliers can detect the boundary points with reasonable accuracy.

For synthetic 2-d and high-dimensional datasets $\alpha = 80\%$ and $\alpha = 60\%$, respectively. For 2-d data, we further checked if the ground truth boundary points cover the boundary regions of the cluster by visual inspection. In real data, we considered $\alpha = 80\%$ for Biomed [10] and Cancer [11] datasets, respectively.

We compared the accuracy of *StaticBPF* with BorderShift [9], BPDAD [27], BRIM [35] and BORDER [41]. For all methods, we chose the parameter value ranges as shown in Table 2. BPDAD uses fixed parameter values and automatically outputs the boundary points. Therefore, we mention the number of boundary points output by BPDAD for all datasets. For BRIM, there are no suggested range for *eps* which is a distance parameter and the points within *eps* distance are considered as the neighborhood of a target point. We considered 5 values for *eps* within the range of the minimum and average distance between all points in the given dataset. For BorderShift, we tuned its k parameter according to the suggested range. However, tuning its λ_1 and λ_2 parameters requires prior knowledge of the number of outliers in the dataset. Since in our experiments the number of outliers are known, we show the results of BorderShift for the best λ_1 and λ_2. However, without the knowledge of the number of outliers in the dataset, tuning λ_1 and λ_2 can be challenging.

For quantitative evaluation, precision, recall, F1 score, area under ROC curve (AUC ROC) and area under precision-recall curve (AUC PR) are used. AUC ROC and AUC PR are affected by ranking of points w.r.t. their scores. In BorderShift, λ_1 and λ_2 are start and end pointers, respectively, to the boundary points occurring in the list of points sorted in ascending order of the scores. We considered the ranking from λ_2 until the start of the list (backwards) to calculate AUC ROC and AUC PR. This ranking is reasonable as, given that λ_1 and λ_2 are tuned, points occurring before λ_1 are likely nearer to the core points. Similarly, points after λ_2 are likely farther from the core points or can be outliers. For *StaticBPF*, BORDER and BRIM, we considered ranking of points according to the calculated scores. AUC ROC and AUC PR for BPDAD cannot be included as its output is not based on ranking.

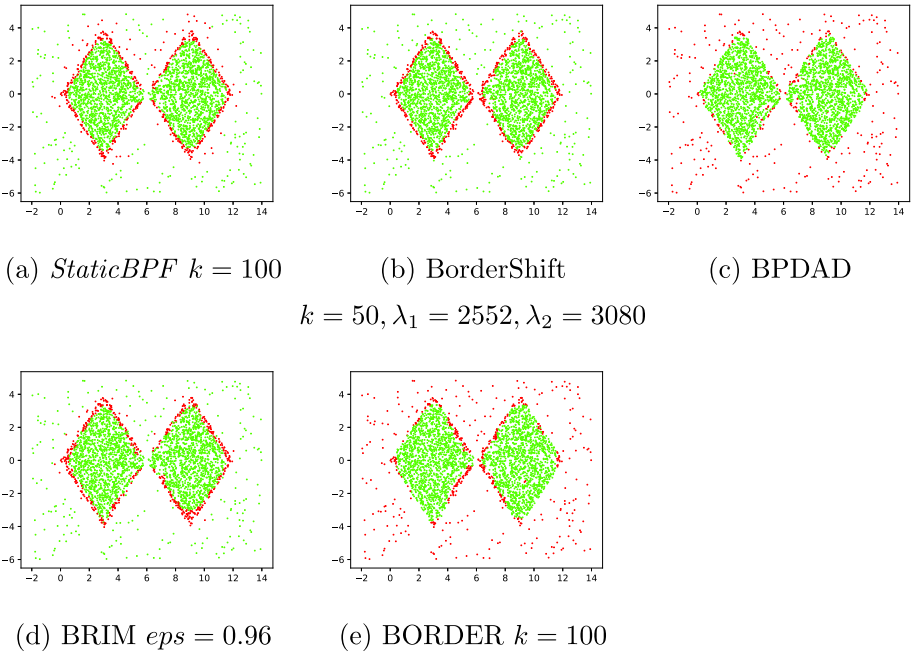
5.3.2 2-Dimensional synthetic data

The evaluation of *StaticBPF* on 2-d synthetic datasets are of two types: (1) visual demonstration of the detected boundary points, and (2) quantitative evaluation of accuracy in terms of various metrics. The details of 2-d datasets are given in Table 3.

Figures 2, 3, 4 and 5 show the results of the top-m boundary points (red points) detected by all methods and their corresponding quantitative accuracy in Tables 4, 5, 6 and 7 at the optimal parameter values where the bold numbers represent the best results.

Table 3 Details of 2- and high-dimensional synthetic datasets

Dataset name	Dataset size (n)	#outliers	#boundary points (m)
Diamonds	3300	220	528
Rings	4200	172	567
Mix1	3800	262	838
Mix2	1710	47	479
Mix3	1800	127	484
Mix4	2400	200	298
10d	3300	372	939
20d	3300	401	997
50d	3300	400	1140



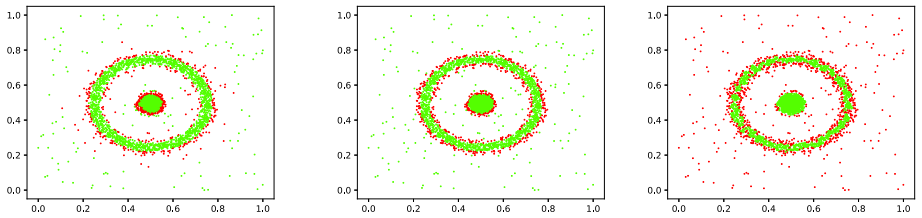
(a) *StaticBPF* $k = 100$ (b) BorderShift $k = 50, \lambda_1 = 2552, \lambda_2 = 3080$ (c) BPDAD

(d) BRIM $eps = 0.96$ (e) BORDER $k = 100$

Fig. 2 Diamonds dataset: $n = 3300$, #*outliers* $= 220$, $m = 528$

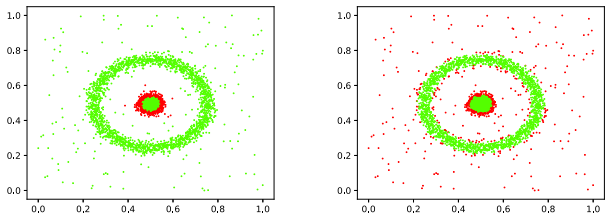
Table 4 Accuracy on diamonds dataset

Method	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 100$	0.75	0.75	0.75	0.96	0.75
BPDAD	#boundaries = 362	0.33	0.23	0.27	–	–
BRIM	$eps = 0.96$	0.71	0.71	0.71	0.95	0.75
BORDER	$k = 100$	0.55	0.55	0.55	0.86	0.41
BorderShift	$k = 50$	**0.83**	**0.83**	**0.83**	**0.97**	**0.85**
	$\lambda_1 = 2552, \lambda_2 = 3080$					



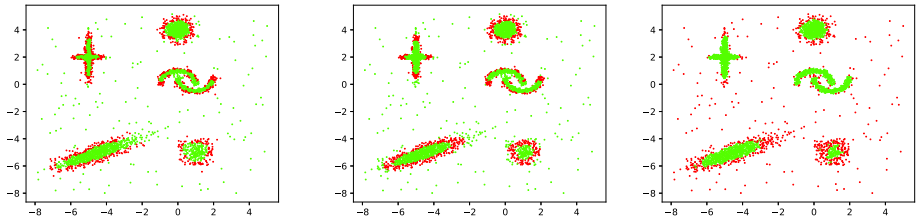
(a) *StaticBPF* $k = 200$ (b) BorderShift (c) BPDAD

$k = 100, \lambda_1 = 3461, \lambda_2 = 4028$



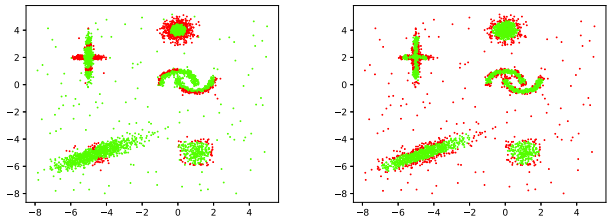
(d) BRIM $eps = 0.094$ (e) BORDER $k = 100$

Fig. 3 Rings dataset: $n = 4200$, #*outliers* $= 172$, $m = 567$



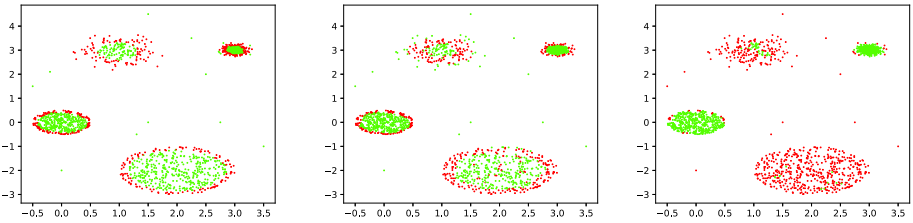
(a) *StaticBPF* $k = 100$ (b) BorderShift (c) BPDAD

$k = 100, \lambda_1 = 2700, \lambda_2 = 3538$



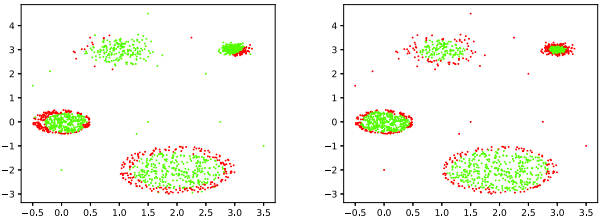
(d) BRIM $eps = 1.13$ (e) BORDER $k = 50$

Fig. 4 Mix1 dataset: $n = 3800$, #*outliers* $= 262$, $m = 838$



(a) *StaticBPF* $k = 100$ (b) BorderShift (c) BPDAD

$k = 50, \lambda_1 = 1184, \lambda_2 = 1663$



(d) BRIM $eps = 1.196$ (e) BORDER $k = 100$

Fig. 5 Mix2 dataset $n = 1710$, $\#outliers = 47$, $m = 479$

Table 5 Accuracy on rings dataset

Method	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 200$	**0.79**	**0.79**	**0.79**	**0.97**	**0.71**
BPDAD	#boundaries = 915	0.33	0.53	0.4	–	–
BRIM	$eps = 0.094$	0.53	0.53	0.53	0.78	0.59
BORDER	$k = 100$	0.69	0.69	0.69	0.95	0.55
BorderShift	$k = 100$	0.66	0.66	0.66	0.93	0.69
	$\lambda_1 = 3461, \lambda_2 = 4028$					

Table 6 Accuracy on Mix1 dataset

Method	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 100$	**0.72**	**0.72**	**0.72**	**0.91**	**0.73**
BPDAD	#boundaries = 716	0.49	0.42	0.45	–	–
BRIM	$eps = 1.13$	0.39	0.39	0.39	0.69	0.45
BORDER	$k = 50$	0.65	0.65	0.65	0.86	0.51
BorderShift	$k = 100$	0.69	0.69	0.69	0.88	0.69
	$\lambda_1 = 2700, \lambda_2 = 3538$					

Table 7 Accuracy on Mix2 dataset

Method	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 100$	0.85	0.85	0.85	**0.96**	**0.81**
BPDAD	#boundaries = 689	0.32	0.46	0.38	–	–
BRIM	$eps = 1.19$	0.63	0.63	0.63	0.75	0.69
BORDER	$k = 100$	**0.9**	**0.9**	**0.9**	**0.96**	0.79
BorderShift	$k = 50$	0.71	0.71	0.71	0.89	0.76
	$\lambda_1 = 1184, \lambda_2 = 1663$					

Table 8 Accuracy on Mix3 dataset

Method	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 100$	**0.84**	**0.84**	**0.84**	**0.96**	**0.82**
BPDAD	#boundaries = 453	0.35	0.33	0.34	–	–
BRIM	$eps = 1.2$	0.65	0.65	0.65	0.74	0.65
BORDER	$k = 100$	0.71	0.71	0.71	0.89	0.58
BorderShift	$k = 50$	0.72	0.72	0.72	0.9	0.79
	$\lambda_1 = 1189, \lambda_2 = 1673$					

Figure 2 shows the results on Diamonds dataset (used in [9, 35, 41]). *StaticBPF* and BorderShift can clearly detect the boundary points while other methods are not as accurate. However, few outliers are also detected as boundary points by *StaticBPF* but most of the points lie on the clear boundaries of the shapes. Referring to the quantitative accuracy shown in Table 4, *StaticBPF* shows comparable performance to BorderShift. However, tuning the best parameter of BorderShift is a difficult task if the number of outliers are not known. Moreover, BRIM shows a comparable accuracy to *StaticBPF* and BorderShift while the accuracy of BORDER and BPDAD are influenced by outliers. On rings [34] dataset (Fig. 3 and Table 5), *StaticBPF* performs better than other methods.

Mix1, Mix2 and Mix3 dataset consists of shapes having different number of points. Mix2 and Mix3 are similar to the dataset used in the LOF paper [5] with clusters having points from uniform and normal distribution. However, the difference between Mix2 and Mix3 is the number of outliers. The purpose of using these datasets is to demonstrate the robustness of *StaticBPF* on datasets having clusters of arbitrary shapes and densities. On Mix1 (Fig. 4 and Table 6), *StaticBPF* performs better than other methods, while BorderShift shows comparable performance. In Mix2 (Fig. 5 and Table 7), BORDER performs better than *StaticBPF* in terms of precision, recall and F1 score. This is due to the presence of small number of outliers (only 19). However, when we increased the number of outliers in Mix3 (84 outliers), the performance of BORDER dropped due to the presence of outliers, whereas *StaticBPF* showed almost consistent performance as shown in Table 8. We omit the boundary points detected by all methods on Mix3 dataset to avoid redundancy. The results on Mix2 and Mix3 suggest that *StaticBPF* is robust to outliers. For other methods, the parameters were adjusted to give consistent results.

On Mix4 dataset, the ground truth is obtained by considering the shape of the clusters. Mix4 dataset has two circular clusters of uniformly distributed points with different radii and number of points. The small and large clusters have radius $r_1 = 1$ and $r_2 = 2$ with 1400 and

Table 9 Accuracy comparison on Mix4 dataset with ground truth obtained with BORDER

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 50$	**0.74**	**0.74**	**0.74**	**0.97**	**0.78**
	$k = 100$	0.72	0.72	0.72	**0.97**	0.76
	$k = 150$	0.67	0.67	0.67	**0.97**	0.72
BorderShift	$k = 50, \lambda_1 = 2102, \lambda_2 = 2400$	0.33	0.33	0.33	0.86	0.31
	$k = 50, \lambda_1 = 1902, \lambda_2 = 2200$	0.6	0.6	0.6	0.91	0.67
	$k = 50, \lambda_1 = 1702, \lambda_2 = 2000$	0.42	0.42	0.42	0.55	0.25
BRIM	$eps = 0.58$	0.52	0.52	0.52	0.82	0.57
	$eps = 1.15$	0.56	0.56	0.56	0.76	0.53
	$eps = 1.73$	0.39	0.39	0.39	0.76	0.5
BORDER	$k = 100$	0.32	0.32	0.32	0.82	0.29
	$k = 150$	0.35	0.35	0.35	0.86	0.32
	$k = 200$	0.37	0.37	0.37	0.87	0.34
BPDAD	#boundary pts = 740	0.15	0.37	0.21	–	–

Table 10 Accuracy comparison on Mix4 dataset with ground truth obtained by using radii of the circular clusters

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 50$	**0.75**	**0.75**	**0.75**	**0.97**	**0.8**
	$k = 100$	**0.75**	**0.75**	**0.75**	**0.97**	**0.8**
	$k = 150$	0.7	0.7	0.7	**0.97**	0.76
BorderShift	$k = 50, \lambda_1 = 2010, \lambda_2 = 2300$	0.49	0.49	0.49	0.92	0.44
	$k = 50, \lambda_1 = 1810, \lambda_2 = 2100$	0.48	0.48	0.48	0.63	0.31
	$k = 50, \lambda_1 = 1610, \lambda_2 = 1900$	0.25	0.25	0.25	0.3	0.12
BRIM	$eps = 0.74$	0.46	0.46	0.46	0.76	0.53
	$eps = 1.4$	0.44	0.44	0.44	0.74	0.38
	$eps = 2.16$	0.25	0.25	0.25	0.68	0.26
BORDER	$k = 50$	0.33	0.33	0.33	0.74	0.24
	$k = 100$	0.31	0.31	0.31	0.81	0.29
	$k = 150$	0.33	0.33	0.33	0.85	0.31
BPDAD	#boundary pts = 740	0.16	0.42	0.23	–	–

800 points, respectively. We obtained the boundary points ground truth using two methods: (i) applying BORDER (as explained in Sect. 5.3.1), and (ii) adjust β to cover the region close to the boundary of the circular clusters where the points within $r_1 - (r_1 - \beta)$ and $r_2 - (r_2 - \beta)$ in the small and large clusters, respectively, are considered boundary points of the circular clusters. $\beta = 0.05$ for small and $\beta = 0.2$ for large cluster. The results are in Tables 9 and 10 for the ground truth obtained via BORDER and by adjusting β, respectively. Moreover, we show the change in accuracy w.r.t. the change in parameters where the best and sub-optimal results are shown for each method.

Table 11 Comparison of accuracy on 10-dimensional dataset

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 150$	0.78	0.78	0.78	**0.94**	0.78
	$k = 200$	0.79	0.79	0.79	**0.94**	0.79
	$k = 250$	**0.8**	**0.8**	**0.8**	**0.94**	**0.8**
BorderShift	$k = 100, \lambda_1 = 2189, \lambda_2 = 3128$	0.54	0.54	0.54	0.74	0.61
	$k = 100, \lambda_1 = 1989, \lambda_2 = 2928$	0.53	0.53	0.53	0.62	0.43
	$k = 100, \lambda_1 = 1789, \lambda_2 = 2728$	0.36	0.36	0.36	0.48	0.21
BRIM	$eps = 0.38$	0.5	0.5	0.5	0.69	0.54
	$eps = 0.58$	0.62	0.62	0.62	0.79	0.62
	$eps = 0.77$	0.43	0.43	0.43	0.59	0.4
BORDER	$k = 100$	0.58	0.58	0.58	0.83	0.49
	$k = 150$	0.59	0.59	0.59	0.83	0.49
	$k = 200$	0.6	0.6	0.6	0.83	0.49
BPDAD	#boundary pts = 872	0.23	0.21	0.22	–	–

Referring to the overall results in Tables 9 and 10, *StaticBPF* outperforms all methods and shows almost consistent accuracy on different k values. Similarly, accuracy of BORDER is also consistent with changing k. On the other hand, accuracy of BRIM and BorderShift changes with the change in *eps*, and λ_1 and λ_2 at fixed k, respectively. λ_1 and λ_2 cover the top m points and their values were changed with the interval of 200 points. Hence, the results suggest that tuning λ_1 and λ_2 is difficult when the number of outliers is not known. Also, the ground truth obtained via BORDER is reasonable as the accuracy does not change significantly when we consider the shapes of clusters to obtain the ground truth.

Furthermore, on Mix4 dataset we show that the k value tuned for *StaticBPF* for boundary points detection can be used with LOF for outlier detection. LOF showed a consistent accuracy w.r.t. precision, recall and F1 score of 0.94 for detecting 200 outliers at $k = 50, 100, 150$. This shows that LOF and *StaticBPF* can work with the same k value.

5.3.3 High-dimensional synthetic data

This subsection shows the results of accuracy evaluation on synthetic high-dimensional data of dimensionality 10, 20 and 50. In order to obtain the clusters of different sizes and densities, the generated high-dimensional data have three spherical clusters of different radii with data points from Gaussian distribution. The details are shown in Table 3. The results are shown in Tables 11, 12 and 13 and the accuracy is shown for different parameter values to show the change in accuracy w.r.t. change in parameters.

Overall, *StaticBPF* performs consistently better than other methods on high dimensional data. It can be observed that larger k gives better results for *StaticBPF* according to all metrics. On the other hand, BORDER shows a consistent accuracy upon increasing k on all dimensionality. However, BorderShift struggles to detect the boundary points in high-dimensional data and choosing appropriate λ_1 and λ_2 affects its accuracy. Similarly, tuning *eps* of BRIM is also difficult as the accuracy changes with the change in *eps*.

Table 12 Comparison of accuracy on 20-dimensional dataset

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 100$	0.7	0.7	0.7	0.82	0.69
	$k = 200$	0.72	0.72	0.72	**0.83**	0.71
	$k = 250$	**0.74**	**0.74**	**0.74**	**0.83**	**0.83**
BorderShift	$k = 100, \lambda_1 = 1902, \lambda_2 = 2899$	0.47	0.47	0.47	0.64	0.37
	$k = 100, \lambda_1 = 1702, \lambda_2 = 2699$	0.37	0.37	0.37	0.69	0.45
	$k = 100, \lambda_1 = 1502, \lambda_2 = 2499$	0.36	0.36	0.36	0.67	0.51
BRIM	$eps = 0.55$	0.59	0.59	0.59	0.76	0.63
	$eps = 0.82$	0.62	0.62	0.62	0.78	0.66
	$eps = 1.1$	0.41	0.41	0.41	0.63	0.47
BORDER	$k = 100$	0.57	0.57	0.57	0.72	0.44
	$k = 150$	0.59	0.59	0.59	0.72	0.45
	$k = 200$	0.61	0.61	0.61	0.75	0.42
BPDAD	#boundary pts = 1033	0.22	0.23	0.22	–	–

Table 13 Comparison of accuracy on 50-dimensional dataset

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 150$	0.81	0.81	0.81	0.93	**0.78**
	$k = 200$	0.83	0.83	0.83	**0.94**	**0.78**
	$k = 250$	**0.84**	**0.84**	**0.84**	**0.94**	**0.78**
BorderShift	$k = 100, \lambda_1 = 1760, \lambda_2 = 2900$	0.69	0.69	0.69	0.71	0.49
	$k = 100, \lambda_1 = 1560, \lambda_2 = 2700$	0.74	0.74	0.74	0.78	0.65
	$k = 100, \lambda_1 = 1360, \lambda_2 = 2500$	0.68	0.68	0.68	0.78	0.75
BRIM	$eps = 1.2$	0.55	0.55	0.55	0.79	0.72
	$eps = 1.72$	0.48	0.48	0.48	0.71	0.53
	$eps = 2.15$	0.36	0.36	0.36	0.53	0.37
BORDER	$k = 100$	0.65	0.65	0.65	0.81	0.53
	$k = 150$	0.65	0.65	0.65	0.81	0.52
	$k = 200$	0.65	0.65	0.65	0.81	0.53
BPDAD	#boundary pts = 1100	0.23	0.22	0.22	–	–

Furthermore, we measured the outlier detection accuracy of LOF in terms of precision, recall and F1 score on high dimensional datasets. On 10-d dataset, LOF's accuracy is 0.82 for $k = 150, 200, 250$. Moreover, on 20-d dataset the accuracy is 0.79, 0.78, 0.77 for $k = 100, 200, 250$, respectively. On 50-d dataset, the accuracy is 0.79 for $k = 150, 200, 250$. Hence, the results suggest that k appropriate for *BPF* for boundary points detection can be used with *LOF* for outlier detection on high-dimensional datasets.

Table 14 Details of real datasets for quantitative accuracy evaluation

Dataset name	Dataset size (n)	Dimensions (d)	#Outliers	#Boundaries (m)
Biomed [10]	194	4	14	42
Cancer [11]	449	9	88	129

Table 15 Accuracy on Biomed dataset

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 150$	0.64	0.64	0.64	0.92	0.73
	$k = 160$	0.69	0.69	0.69	0.95	0.79
	$k = 170$	**0.86**	**0.86**	**0.86**	**0.97**	**0.95**
BorderShift	$k = 100, \lambda_1 = 138, \lambda_2 = 180$	0.83	0.83	0.83	0.96	0.88
	$k = 100, \lambda_1 = 118, \lambda_2 = 160$	0.52	0.52	0.52	0.59	0.42
	$k = 100, \lambda_1 = 98, \lambda_2 = 140$	0.14	0.14	0.14	0.18	0.034
BRIM	$eps = 0.54$	0.64	0.64	0.64	0.91	0.69
	$eps = 0.6$	0.69	0.69	0.60	0.91	0.76
	$eps = 0.64$	0.67	0.67	0.67	0.91	0.75
BORDER	$k = 150$	0.67	0.67	0.67	0.91	0.56
	$k = 160$	0.69	0.69	0.69	0.91	0.55
	$k = 170$	0.71	0.71	0.71	0.82	0.51
BPDAD	#boundary pts = 49	0.53	0.62	0.57	–	–

5.3.4 Real data

This section presents the results of evaluating accuracy on real datasets. Unlike outlier detection, there are no benchmark real datasets available for verifying the performance of boundary point detection. Therefore, we use the same datasets used in the related work for evaluation.

Firstly, we show the accuracy of *StaticBPF* and its competitors quantitatively on real datasets Biomed [10] and Cancer [11] as used in [6, 9, 34]. The ground truth labels of boundary points are not available for these datasets. Therefore, we applied the method explained in Sect. 5.3.1 to obtain the ground truth. Moreover, the duplicate points are removed from Cancer data prior to ground truth extraction.

Table 14shows the details of these datasets. Similar to the synthetic dataset, we show three measurements for three parameter values for all methods to show the fluctuation in accuracy w.r.t. parameter values. We tuned BORDER and *StaticBPF* in the range [50, 190] with interval of 10 on Biomed dataset. On Cancer dataset, the range of $k = [50, 250]$ with interval of 10 for both methods. The results in Tables Tables 15 and 16show that StaticBPF has comparable or better accuracy than its competitors on Biomed and Cancer datasets. Moreover, our method shows better accuracy with larger k and a similar trend may be observed with BORDER. However, BorderShift's accuracy depends on how λ_1 and λ_2 are tuned for the fixed k.

The outlier detection accuracy of LOF in terms of precision, recall and F1 score on Biomed data is 0.86 on $k = 150, 160, 170$. On Cancer data, the accuracy of LOF is 0.37, 0.43 and 0.57 for $k = 190, 200, 210$, respectively.

Secondly, we demonstrate the accuracy of *StaticBPF* visually on MNIST and ORL datasets as used in [9, 34]. For both datasets, we show the BPF and LOF scores respectively, at the

Table 16 Accuracy on cancer dataset

Methods	Parameter	Prec.	Rec.	F1	ROC	PR
StaticBPF	$k = 190$	0.53	0.53	0.53	0.75	0.69
	$k = 200$	**0.59**	**0.59**	**0.59**	0.79	0.72
	$k = 210$	**0.59**	**0.59**	**0.59**	**0.83**	**0.73**
BorderShift	$k = 50, \lambda_1 = 232, \lambda_2 = 361$	0.26	0.26	0.26	0.76	0.59
	$k = 50, \lambda_1 = 212, \lambda_2 = 341$	0.3	0.3	0.3	0.7	0.56
	$k = 50, \lambda_1 = 192, \lambda_2 = 321$	0.21	0.21	0.21	0.59	0.41
BRIM	$eps = 0.75$	0.49	0.49	0.49	0.66	0.5
	$eps = 1$	0.51	0.51	0.51	0.75	0.52
	$eps = 1.25$	0.46	0.46	0.46	0.64	0.49
BORDER	$k = 90$	0.54	0.54	0.54	0.81	0.56
	$k = 100$	0.53	0.53	0.53	0.82	0.56
	$k = 110$	0.52	0.52	0.52	0.82	0.56
BPDAD	#boundary pts = 244	0.29	0.54	0.37	–	–

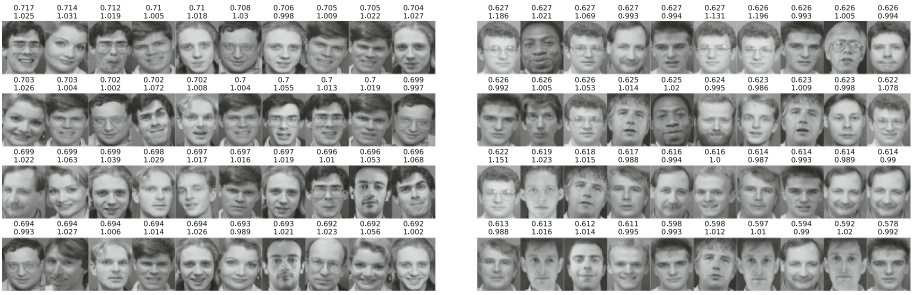


Fig. 6 **a** Top and **b** bottom 40 faces detected by *StaticBPF* ($k = 50$) labeled with BPF and LOF scores

top of each image. Similar to [9, 34], the goal of this experiment is to show the top boundary points w.r.t. the BPF scores.

The Olivetti Research Laboratory (ORL) face dataset [4] consists of 400 images of 92×112 pixels of frontal faces of 40 people where each pixel represents the gray value in the range of 0–255. The images of frontal faces are considered as core images, while the images with left and right profile faces are considered as the boundary images. Each image is transformed from 92×112 to $1 \times 10{,}304$ by concatenating each subsequent row to the previous row of the image. Figure 6a, b shows the top 40 images with largest and smallest BPF scores, respectively. It may be observed that majority of the top 40 faces are the non-frontal images. The bottom 40 images are shown for comparison between frontal and non-frontal images.

We further showed the effectiveness of *StaticBPF* visually on MNIST [25] dataset of handwritten digits as used in [9, 34]. The dataset contains 60,000 images in training and 10,000 images in testing set of 28×28 pixels. We selected digit '3' from testing set of MNIST dataset. It contains 1010 images of digit '3' which are considered as core and boundary images. The easily recognizable 3s can be regarded as cores and distorted 3s can be boundary images. We also selected 100 random images from digits 0, 2, 4, 6, 7 and 9 from testing set

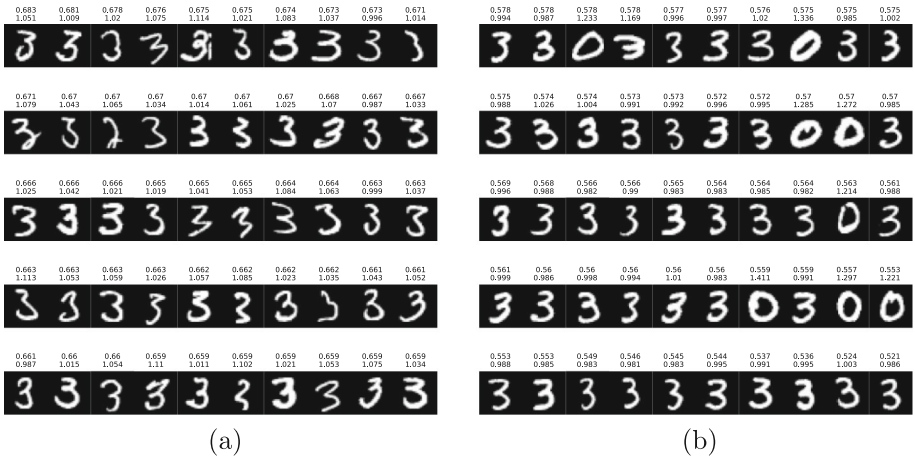


Fig. 7 **a** Top and **b** bottom 50 digits detected by *StaticBPF* ($k = 50$) labeled with BPF and LOF scores

which may be considered outliers. The preprocessing is performed in the same way as done for the ORL dataset.

Figure 7a shows the top 50 boundary points detected by *StaticBPF* having larger BPF scores. The boundary images are distorted 3s and they are ranked higher, while core and outlier images are assigned smaller BPF scores. Consequently, as can be seen in Fig. 7b, the bottom 50 digits are a mix of core and outliers digits. Hence, *StaticBPF* can discriminate boundary points from core points and outliers based on the BPF scores. It can be observed that LOF scores of core and boundary points are close to 1 and outliers have LOF scores > 1.

6 *StreamBPF*

Applying *StaticBPF* on data stream can be computationally expensive. To address this problem, this section presents *StreamBPF*. Firstly, the problem of detecting boundary points in streaming data is defined and then the proposed method is explained.

6.1 Problem definition

Data streams and sliding windows are defined in the following definitions.

Definition 6 (*Data stream*) A data stream is defined as an unbounded sequence of d-dimensional data points $p_1, p_2, \ldots, p_t, p_{t+1}, \ldots$, where $p_t \in \mathbf{R}^d$.

In order to bound the unbounded data stream, we define a count-based sliding window which maintains a fixed number of the recent points (n).

Definition 7 (*Count-based sliding window*) A count-based sliding window W_t of size n at time step t is a set of points $p_{t-n+1}, p_{t-n+2}, \ldots, p_{t-1}, p_t$.

The proposed method is not restricted to the count-based sliding window and other types of window may be used. However, for simplicity, we focus on a count-based sliding window of a fixed size n which slides at every time step. Hence, the problem of detecting boundary points in streaming data can be defined as follows.

Definition 8 (*Boundary points detection in data stream*) Given a window of size n which slides at every time step and a parameter m, boundary points detection over the data stream is the problem of identifying the top-m boundary points having the largest BPF scores in the current window W_t whenever the window slides.

At time step t, given W_t and a new arrival point p, we need to calculate $BPF(p)$ and update BPF scores of each existing point $q \in W_t \backslash \{p\}$. $BPF(p)$ can be calculated from scratch by calculating $N_k(p)$, $G(p)$ and $LOF(p)$. Similarly for each remaining point $q \in W_t \backslash \{p\}$, $G(q)$, $LOF(q)$ and $BPF(q)$ are calculated. A naïve approach is to calculate the BPF scores of all points in W_t from scratch and repeat this computation for each window slide. However, this approach is computationally expensive and not tolerable for streaming data.

To drastically improve the overall runtime performance, *StreamBPF* employs (1) grid-based k-nearest neighbors computation and (2) incremental computation. We discuss the grid-based approach in Sect. 6.2 and present the incremental computation in Sect. 6.3. *StreamBPF* is presented in Sect. 6.4.

6.2 Grid structure

We propose a grid structure to improve the runtime performance of the k-nearest neighbors computation which can contribute to overall improvement of the runtime performance. The following subsections present the definitions and algorithm.

6.2.1 Definitions

The *grid* is a cell-based structure that divides a multi-dimensional data space into cells of equal size. Consider that data in d-dimensional space is normalized in the range $[0, 1]$ in each dimension. The grid cell length can be defined as follows:

Definition 9 (*Cell length*) *Cell Length* l is the length of each side of a d-dimensional cell.

The value of l determines the resolution of the grid and it can affect the runtime performance of our proposed method. Given an appropriate value of l, each point in the window can be assigned a specific cell identified by a unique cell key.

Definition 10 (*Cell key*) Given a point $p = (p_1, p_2, \ldots, p_d) \in W_t$ and cell length l, p belongs to a cell addressed by a d-dimensional key called *Cell key* denoted as K where:

$$K = \left(\left\lfloor \frac{p_1}{l} \right\rfloor, \left\lfloor \frac{p_2}{l} \right\rfloor, \ldots, \left\lfloor \frac{p_d}{l} \right\rfloor \right). \tag{9}$$

Hence, the cell key K is actually the address of a cell in the grid which groups a subset $\mathbb{S}_i$ of points in W_t. Next, the grid of cells can be defined as the following.

Definition 11 (*Grid*) Given a window W_t at time step t, the grid $\mathbb{G}_t$ can be given as:

$$\mathbb{G}_t = \{C_i = (K_i, \mathbb{S}_i) \mid \mathbb{S}_i \neq \{\}, i = 1, 2, 3 \ldots\}, \tag{10}$$

where K_i is a d-dimensional cell key of the ith cell containing a subset $\mathbb{S}_i$ of points in W_t.

The proposed method maintains the grid structure $\mathbb{G}_t$ where a cell $C_i \in \mathbb{G}_t$ exists only if it contains at least one point. Therefore, depending on the cell length l, $\mathbb{G}_t$ will contain no more than n cells. Basically, the proposed grid structure considers only those cells that are populated by points and the maximum number of cells will not exceed n. The number of points in a cell C_i is represented as $|\mathbb{S}_i|$. Next, the definition of minimum distance (*MinDist*) is given as follows.

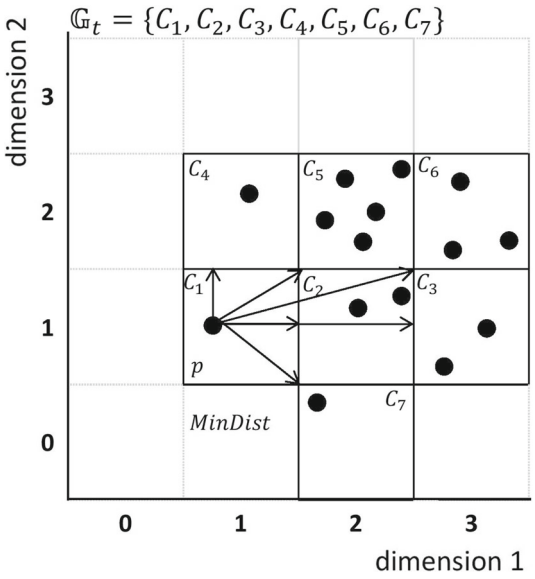


Fig. 8 Example 2-dimensional grid with each dimension partitioned into equal number of bins of equal length

Definition 12 (*MinDist*) Given the coordinates of the bottom-left and top-right corners of a cell C_i $X = (x_1, x_2, \ldots, x_d)$ and $Y = (y_1, y_2, \ldots, y_d)$, respectively, and a point $p = (p_1, p_2, \ldots, p_d)$, the *MinDist* between p and C_i $MinDist(p, C_i)$ can be defined as:

$$MinDist(p, C_i) = \sqrt{\sum_{j=1}^{d} |p_j - a_j|^2} \tag{11}$$

where,

$$a_j = \begin{cases} x_j & p_j < x_j \\ y_j & p_j > y_j \\ p_j & \text{otherwise.} \end{cases} \tag{12}$$

The $MinDist(p, C_i)$ is the distance between a point p and the closest edge of the cell represented by its cell key C_i. This is illustrated in Fig. 8 as an example of 2-dimensional data mapped onto grid $\mathbb{G}_t$. $\mathbb{G}_t$ contains the populated cells $(C_1, C_2, C_3, C_4, C_5, C_6, C_7)$ and no other cells need to be initialized or stored. The point p is mapped to the cell $C_1 = (1, 1)$ and its *MinDist* is represented as the length of arrows originating from p to the closest edges of the nearby cells.

6.2.2 Grid-based *k*-nearest neighbor computation algorithm

A naïve way of calculating the k-nearest neighbors of a point $p \in W_t$ is to consider the pairwise distances $dist(p, q)$ (where $q \in W_t \backslash \{p\}$) with $n - 1$ points in W_t. Consequently, $n-1$ points have to be scanned to calculate distances, sorting them in the ascending order and then obtaining the k-nearest points. However, the proposed grid structure groups all points in cells by partitioning the space where the number of cells is much smaller than n.

Consider the example in Fig. 8 where the target point p belongs to the cell $C_1 = (1, 1)$ for which k-nearest neighbors have to be calculated where $k = 5$. *MinDist* from p and other

cells (excluding C_1) can be calculated according to Definition 12. The cells are sorted in the ascending order of *MinDist* and the number of points are summed for each cell till k or more points are collected. For example, the cells shown in Fig. 8 sorted by *MinDist* from p are given as $C_4, C_2, C_5, C_7, C_3, C_6$ and the 5th neighbor of p lies in C_5 ($|\mathbb{S}_4| + |\mathbb{S}_2| + |\mathbb{S}_5| = 8$). Next, the distance from p and all points in C_4, C_2 and C_5 are calculated and sorted in the ascending order. The distance from p to the 5th nearest neighbor is set to *current_kdist*(p). We need to note that *current_kdist*(p) is not necessarily *kdist*(p). For example, the point in cell C_7 is within the 5 nearest neighbor of p, but C_7 is at a larger *MinDist* value than C_4, C_2 and C_5. To address this, referring to the example in Fig. 8, we need to check the remaining cells (i.e. C_3, C_6, C_7) whose *MinDist*s from p are within *current_kdist*(p). As *current_kdist*(p) is the maximum possible value of $kdist$, we need to check the cells which lie under this *current_kdist*(p). If any cell $C_i \in \mathbb{G}_t \backslash \{C_1, C_2, C_4, C_5\}$ has $MinDist(p, C_i) <$ *current_kdist*(p), then we calculate the distances from p and the points in C_i and update *current_kdist*(p). Therefore, we continue this process until a cell appears that has *MinDist* greater than *current_kdist*(p), and consider *current_kdist*(p) as *kdist*(p). Hence, k-nearest neighbors of p can be obtained in this manner.

The proposed grid structure can significantly reduce the time complexity. The grid uses *MinDist* to each cell $C_i \in \mathbb{G}_t$ instead of n points in the window where number of cells in the grid ($|\mathbb{G}_t|$) can be much smaller than n ($|\mathbb{G}_t| \ll n$), provided that an appropriate cell length l is chosen. In addition, $MinDist(p, C_i) <$ *current_kdist*(p) does not hold for most of the cells, which means that we do not need to check points inside these cells.

Algorithm 2 represents the grid-based k-nearest neighbor computation method. Given an input point p, the algorithm obtains the cell key K_j of the target point p using Definition 10 (line 1). If the cell key already exists in $\mathbb{G}_t$, then p is inserted in that cell, otherwise a new cell is initialized and inserted in $\mathbb{G}_t$ (line 3–8). The *MinDist*s of p with all cells are calculated and they are sorted in the ascending order of *MinDist*s (line 10–12). After that, the sorted cells are traversed and the sum of number of points in each cell (*count*) is obtained. Also, each traversed cell is inserted in $\mathbb{N}$ (line 15–17). If *count* becomes greater than or equal to k (line 18), then the distance of p with each point in cells in $\mathbb{N}$ is calculated (line 19–21). The distances with p are sorted in the ascending order and *current_kdist*(p) is obtained (line 22). Thereafter, the remaining cells that are within *current_kdist*(p) are checked to update *current_kdist*(p). In case, a cell C_i have $MinDist(p, C_i) \leq$ *current_kdist*(p) (line 27), the distances between p and points in $\mathbb{S}_i$ are calculated and *current_kdist*(p) is updated (line 28–30). If a cell is encountered with $MinDist \geq$ *current_kdist*(p), the algorithm stops and no further cells are checked (line 32). Finally, Algorithm 2 returns the k-nearest neighbor of p.

In case a point p expires from W_t, the cell address of p is calculated and it is removed from this cell. If the cell becomes empty after the removal of p then the cell is deleted from $\mathbb{G}_t$.

Another important observation is that the k-neighborhood of a limited points in W_t have to be updated. Refer to Fig. 9a, where $k = 5$ and r is the 5th-neighbor of q. Consider that p arrives in the neighborhood $N_k^{old}(q)$ (indicated by solid circle in Fig. 9a) of an existing point $q \in W_t \backslash \{p\}$. Consequently, $dist(q, p) < kdist(q)$ is true in this case, and therefore the neighborhood can be updated as $N_k^{new}(q)$ where r is removed and o becomes the new kth neighbor of q (indicated by dotted circle in Fig. 9a). On the other hand, no k-neighborhood update is required for all the points $q \in W_t \backslash \{p\}$ which do not satisfy the condition $dist(q, p) < kdist(q)$ (indicated by square points). Therefore, the k-neighborhood of all the points affected by arrival of p can be updated in this way. However, if a point p expires from $N_k(q)$ (as shown in Fig. 9b), then Algorithm 2 is invoked for finding the new k-neighborhood of q. Hence, Algorithm 2 is invoked for a limited number of points in the W_t.

Algorithm 2 *Grid*

Require: Data point p, #neighbors k, Grid cell length l
Obtain the cell key K_j for p using Definition 10.
if $p \notin \mathbb{G}_t$ **then**
  if $K_j \notin \mathbb{G}_t$ **then** /*Initialize and insert a new cell in $\mathbb{G}_t$*/
    Insert p in $\mathbb{S}_j$.
    Insert $C_j = (K_j, \mathbb{S}_j)$ in $\mathbb{G}_t$.
  else/*Insert p in existing cell C_j*/
    Insert p in $\mathbb{S}_j$.
  end if
end if
for $C_i \in \mathbb{G}_t$ **do**
  Insert ($MinDist(p, C_i)$,C_i) in $\mathbb{M}$. /*Calculate *MinDist* using Def.12*/
end for
Sort cells in $\mathbb{M}$ in ascending order of *MinDist*.
Initialize $count \leftarrow 0$.
for $C_i \in \mathbb{M}$ **do** /*Visiting cells sorted w.r.t. *MinDist**/
  $count \leftarrow count + |\mathbb{S}_i|$.
  Insert C_i in $\mathbb{N}$. /* Collecting the visited cells */
  if $k \leq count$ **then**
    for $C_i \in \mathbb{N}$ and $q \in \mathbb{S}_i$ **do**
      Insert $(q, dist(p, q))$ in $\mathbb{L}$.
    end for
    Sort $\mathbb{L}$ by distances in ascending order and obtain $current\_kdist(p)$.
    Break
  end if
end for
for remaining cells $C_i \in \mathbb{M} \setminus \mathbb{N}$ **do**
  if $MinDist(p, C_i) \leq current\_kdist(p)$ **then**
    For all $q \in \mathbb{S}_i$ calculate $(q, dist(p, q))$ and insert in $\mathbb{L}$.
    Sort $\mathbb{L}$ by distances in ascending order.
    Update $current\_kdist(p)$.
  else
    Break /* do not process the remaining cells */
  end if
end for
$\mathbb{K} \leftarrow$ top-k points in $\mathbb{L}$. /*assign k-nearest points in $\mathbb{L}$ to $\mathbb{K}$ */
return $\mathbb{K}$

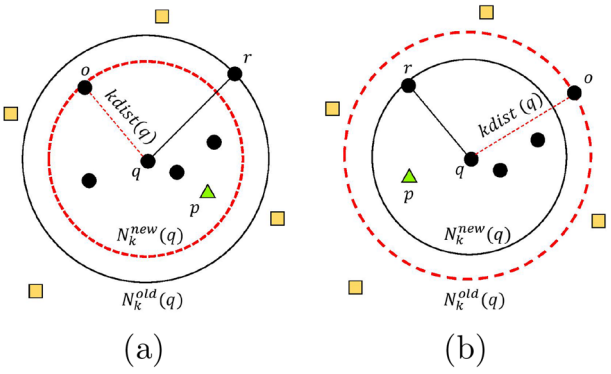


Fig. 9 **a** k-neighborhood affected by the arrival of p. **b** k-neighborhood affected by the expiration of p

6.3 Incremental computation

The BPF scores of a subset of points in W_t are affected by window slide due to the change in their k-neighbors which affects their Gravity values and LOF scores. As a result, the affected points should be identified and their BPF scores should be calculated. In a window of n points, the Gravity values of those points have to be updated whose k-neighborhood have changed due to window slide. However, the window slide affects the LOF scores of a larger number of points.

For a point $q \in W_t$ whose k-neighborhood has changed, the LOF score of q as well as the points which contain q in their k-neighborhood should also be calculated. The detailed explanation and analysis related to the change in LOF scores is presented in [33]. We adopt the same observations in [33] and collect the affected points whose LOF scores should be calculated. Following is the summary of these observations.

- Given that p arrives and r expires from W_t, the k-neighborhood of the points affected by the arrival of p and expiration of r have to be updated. The update of k-neighborhood will result in the update of their *kdist*. Consequently, the update of *kdist* will require the update of $reach\text{-}dist_k$, lrd_k and LOF according to Definitions 1, 2 and 3, respectively. Let $\mathbb{U}$ represent all points whose k-neighborhood have changed, then LOF scores of all points in $\mathbb{U}$ should be updated.
- Given $\mathbb{U}$ the set of points whose k-neighborhood have changed. Let $q \in \mathbb{U}$ and $o \in N_k(q)$, according to Definition 1, if $q \in N_k(o)$, then it will affect the reachability distance of o w.r.t. q ($reach\text{-}dist(o, q)$). Subsequently, the $reach\text{-}dist(o, q)$ will affect the $lrd(o)$ and $LOF(o)$. Therefore, o is also considered as an affected point.
- According to Definition 3, LOF score of a point q should be updated if $lrd(q)$ or any of its k-nearest neighbors $o \in N_k(q)$ changes. If $lrd(o)$ have changed then, all points containing o in their k-nearest neighborhood should be considered affected and their LOF scores should be updated.

The subset of points which have their LOF scores affected due to the window slide can be obtained by the observations given above. Consequently, the BPF scores of all the affected points should be updated.

6.4 Algorithm

This section presents the overall algorithm *StreamBPF* that uses a grid structure to calculate k-nearest neighbors and an incremental method to update BPF scores. The algorithm steps are given as Algorithm 3.

At each time step, *StreamBPF* maintains the k-neighbors of each point as $KNN_t = \{N_k(q) | q \in W_t\}$. Also, it maintains local reachability densities, LOF scores, Gravity values and BPF score as $\mathbb{B}_t = \{(lrd(q), LOF(q), G(q), BPF(q)) \, | q \in W_t\}$. Given a window W_t of size n, a new arrival point p and an expired point r, *StreamBPF* removes r from KNN_t and $\mathbb{B}_t$ (line 1). For the remaining points, which satisfy the condition given on line 4, *StreamBPF* updates the neighborhood of these points (line 5–6). For the points which satisfy condition on line 8, *StreamBPF* invokes Algorithm 2 (line 9). All of those points whose k-neighborhood have been updated are stored in $\mathbb{U}$. The algorithm calculates the Gravity values of all points whose neighborhood have changed (line 14). Next, reverse k-nearest neighbors of all points are calculated (line 16). Thereafter, all the points having LOF scores affected by window slide are collected (line 17–25) and *lrd*s of these points are calculated using Definition 2 (line 26–28). Also, *lrd* of p is calculated (line 29). Finally, the LOF scores, Gravity values and

Algorithm 3 *StreamBPF*

Require: New arrival point p, #neighbors k, #boundary points m
1: Remove $N_k(r)$ from KNN_t and $(lrd(r), LOF(r), G(r), BPF(x))$ from $\mathbb{B}_t$.
2: Obtain $N_k(p) \leftarrow Grid(p, k, l)$ and insert $N_k(p)$ in KNN_t.
3: **for** $q \in W_t \backslash \{p\}$ **do**
4: **if** $dist(q, p) < kdist(q)$ **then** /* point arrives in k-neighbors*/
5: update $N_k(q)$ in KNN_t.
6: Insert q in $\mathbb{U}$.
7: **else**
8: **if** $dist(q, r) \leq kdist(q)$ **then** /* point expires from k-neighbors*/
9: $N_k(q) \leftarrow Grid(q, k, l)$ and update $N_k(q)$ in KNN_t.
10: Insert q in $\mathbb{U}$.
11: **end if**
12: **end if**
13: **end for**
14: Calculate $G(q)$ of all $q \in \mathbb{U}$ and update $G(q)$ in $\mathbb{B}_t$.
15: $\mathbb{U}_{affected} \leftarrow \mathbb{U}$.
16: calculate RN_k reverse k-nearest neighbors of all points in W_t.
17: **for** $q \in \mathbb{U}$ and $o \in N_k(q)$ **do**
18: **if** $q \in N_k(o)$ **then**
19: Insert o in $\mathbb{U}_{affected}$.
20: **end if**
21: **end for**
22: $\mathbb{U}_{update} \leftarrow \mathbb{U}_{affected}$
23: **for** $q \in \mathbb{U}_{affected}$ **do**
24: Insert $RN_k(q)$ in $\mathbb{U}_{update}$.
25: **end for**
26: **for** $q \in \mathbb{U}_{update}$ **do**
27: Calculate $lrd(q)$ and update in $\mathbb{B}_t$.
28: **end for**
29: Calculate $lrd(p)$ and insert in $\mathbb{B}_t$.
30: **for** $q \in \mathbb{U}_{update}$ **do**
31: Calculate $LOF(q)$, $G(q)$, $BPF(q)$ and update in $\mathbb{B}_t$.
32: **end for**
33: Calculate $LOF(p)$, $G(p)$, $BPF(p)$ and insert in $\mathbb{B}_t$.
34: Sort points in $\mathbb{B}_t$ in descending order of BPF scores.
35: $\mathbb{C}_t \leftarrow$ top-m points in $\mathbb{B}_t$.
return $\mathbb{C}_t$

BPF scores of p and the affected points are calculated (line 30–33). The scores are sorted in the descending order of BPF scores and the top-m boundary points in W_t are returned (line 34–35).

The Algorithm 3 considers the arrival and expiration of one point at each time step. However, this algorithm can be easily extended to the case in which more than one points arrive and expire at each time step.

6.5 Evaluation of *StreamBPF*

In this section, the runtime performance evaluation of *StreamBPF* is performed on synthetic and real data. Since the accuracy evaluation *StaticBPF* is already given in Sect. 5.3, we do not further evaluate the accuracy of *StreamBPF* as it computes the exact BPF scores like *StaticBPF*. Moreover, to the best of our knowledge, there are no other methods of boundary points detection for streaming data. Therefore, we compare *StreamBPF* with its variants, as summarized in Table 17, in order to demonstrate the performance improvement achieved due

Table 17 Description of *StreamBPF* and its variants used for runtime evaluation

Method name	Description
StaticBPF	Apply *StaticBPF* (Algorithm 1) on each window
IncBPF	Incrementally computes BPF scores of the affected points in each window
GridBPF	Uses grid for k-neighborhood computation and computes BPF scores of all points in each window
StreamBPF	Apply *StreamBPF* (Algorithm 3) on each window

Table 18 Parameter settings for runtime evaluation

Parameter	Values
#Nearest neighbors (k)	50, **100**, 150, 200, 250
Slide size (w)	**1**, 10, 20, 30, 40, 50
Window size (n)	3000, 6000, **9000**, 12,000, 15,000
Dimensions (d)	**2**, 10, 20, 50, 100

to the proposed grid structure and incremental method.

All algorithms are implemented on Java 1.8 and experiments were executed on workstation with 32GB memory and Intel Core i7-7700 CPU with Windows 10 Pro 64-bits installed.

6.5.1 Synthetic data streams

We evaluated CPU runtime by changing the number of nearest neighbors (k), slide size (w), window size (n) and dimensionality (d) on synthetic data streams. Table 18 shows the range of parameters where the bold and underlined values are default unless otherwise stated. The synthetic data stream consists of 3 Gaussian clusters of different means and standard deviations, and randomly generated outliers. In the initial window W_0, n randomly sampled points from the three Gaussian clusters and outliers are loaded, and their BPF scores are calculated. Thereafter, w oldest points are deleted from the window and w new points are inserted in the window randomly drawn from the same distribution at each time step. The CPU runtime of each window is recorded from W_1 till all points in the data stream have arrived. For all experiments, the window slides 100 times and average CPU runtime is reported in seconds. Moreover, all datasets are normalized in the range [0,1] and the default value of the grid cell size is $l = 0.05$. In the experiments where we evaluated the effect of changing dimensionality, we adjusted the cell length (l) as 0.05, 0.2, 0.45, 0.45 and 0.45 for dimensionality 2, 10, 20, 50 and 100, respectively. The results are shown in Fig. 10.

Figure 10a shows the effect of increasing the slide size (w) on CPU runtime in logarithmic scale. At $w = 1$, *StreamBPF* performs more than 2× faster than *GridBPF* and *IncBPF*, and more than 150× faster than $StaticBPF$. This is because, the expiration and arrival of one point affects the k-nearest neighborhoods and LOF scores of a small fraction of points. Consequently, BPF scores of a small number of points have to be updated. Moreover, the grid structure aids in reducing the runtime of the k-nearest neighbor computation. However, when w increases, the runtime of *StreamBPF* and *GridBPF* becomes comparable as BPF scores of a larger number of points have to be updated. Hence, the incremental computation becomes less useful at larger w. Similarly, the runtime of *IncBPF* increases with w due to

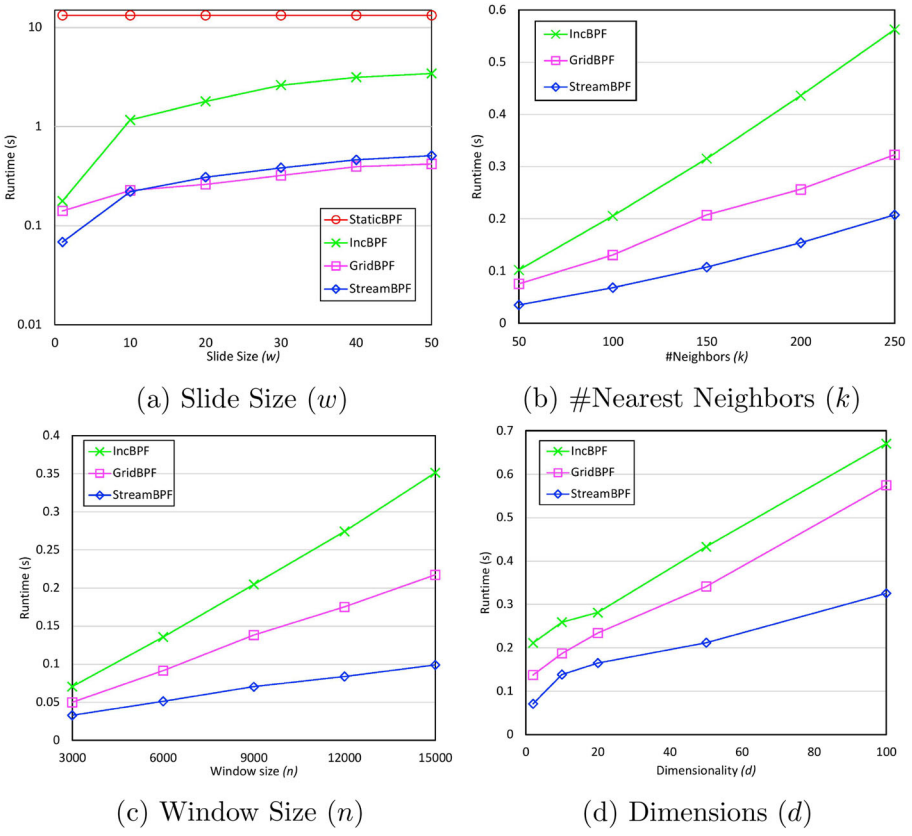


Fig. 10 CPU runtime performance evaluation of the effect of changing parameters

the same reason. However, *StreamBPF* still performs better than *IncBPF* due to the proposed grid structure.

In Fig. 10b–d, the results of *StaticBPF* are not included for clarity and the runtime is shown in linear scale.

Figure 10b shows the impact of increasing the number of nearest neighbors k to calculate BPF scores. Overall, the runtime of all methods increases with k as more points have to be processed to calculate BPF scores. However, *StreamBPF* consistently performs better than other methods due to the proposed grid structure and incremental computation.

Figure 10c, d shows the impact of increasing window size (n) and dimensionality (d), respectively. It may be observed that the runtime of *StreamBPF* improves as parameters n and d increase. This is because when n is increased at fixed k and w, the k-neighborhood and LOF scores of smaller number of points have to be updated in comparison with with the total number of points in the window. Consequently, BPF scores of a small fraction of points have to be updated. Furthermore, *StreamBPF*'s runtime does not increase drastically due to increase in d as the incremental computation avoids redundant computation unlike *GridBPF*.

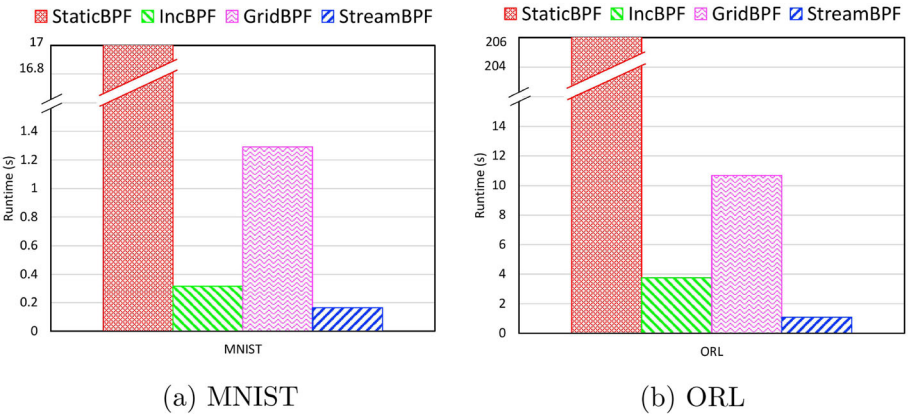


Fig. 11 CPU runtime performance evaluation on real data

6.5.2 Real data streams

For evaluation on real data streams, we simulated the data stream from real datasets MNIST and ORL. Originally, MNIST and ORL datasets have 1010 and 400 data points, respectively. We preprocessed these datasets as explained in Sect. 5.3.4. Hence, the dimensionalities of MNIST and ORL are 784 and 10,304, respectively. We scaled these datasets by random sampling of points where data points were allowed to repeat in order to simulate streaming data. We set window size $n = 3000$ and the grid cell length $l = 0.01$ for MNIST and $l = 0.02$ for ORL. Also, we set $k = 50$ as *StaticBPF* showed reasonable accuracy on ORL and MNIST at this value. The results of runtime evaluation are shown in Fig. 11.

On MNIST, *StreamBPF* performs 8× and 1.9× faster than *GridBPF* and *IncBPF*, respectively. On the other hand, *IncBPF* performs 4× faster than *GridBPF*. On ORL, *StreamBPF* performs 10× and 3× faster than *GridBPF* and *IncBPF*, respectively, while *IncBPF* performs 2.5× faster than *GridBPF*. The performance improvement of *StreamBPF* and *IncBPF* compared with *StaticBPF* and *GridBPF* can be attributed to the incremental computation which allowed the update of BPF scores of the points affected by window slide. As $w = 1$ and $k = 50$, a small fraction of point are affected by the window slide, and therefore smaller number of BPF scores have to be updated. Hence, the incremental computation mainly contributed in the performance improvement of *StreamBPF* and *IncBPF*. The further improvement in *StreamBPF*'s performance is due to the grid structure. Since these data streams contained duplicate points, same points were mapped onto the same cells. As a result, the grid structure used less number of cells to cover all the points in the window. Hence, to calculate the k-nearest neighbors of a target point, lesser number of cells were processed resulting in faster performance as observed in these experiments.

7 Conclusion

In conclusion, this paper has targeted the problem of detecting boundary points from static and streaming data. Firstly, we proposed a boundary points detection method called *BPF* which calculates BPF scores based on Gravity and LOF scores to identify boundary points in a dataset. The boundary points get larger BPF scores than core points and outliers. Based

on *BPF*, we proposed *StaticBPF*, which can detect the top-m boundary points in a static dataset. We evaluated *StaticBPF* on synthetic and real data where we showed its accuracy visually and quantitatively using various metrics. *StaticBPF* showed comparable or better results compared with other methods on synthetic and real datasets. Overall, *StaticBPF* was found robust to the presence of clusters of different shapes, sizes and densities. Furthermore, we showed experimentally that k parameter tuned for *StaticBPF* can be used with LOF for outlier detection.

Secondly, this paper proposed *StreamBPF* for boundary points detection over streaming data. *StreamBPF* employed a grid structure for the k-nearest neighbors computation and an incremental computation method to update BPF scores of the points affected by window slide. The experiment results showed that the proposed grid structure can effectively improve the k-nearest neighbors computation. Also, the incremental computation method was found to be advantageous if the number of affected points due to window slide are small. Overall, the results suggest that *StreamBPF* can be used for boundary points detection in streaming data.

As the future direction, working on an approximate method of detecting boundary points can be an interesting direction to extend this work.

Acknowledgements This work was partly supported by JSPS KAKENHI Grant Numbers JP19H04114, JP22H03694 and JP22K19802, JST CREST Grant Number JP-JCR22M2, NEDO Grant Number JPNP20006, and AMED Grant Number JP21zf0127005.

Funding This work was partly supported by JSPS KAKENHI Grant Numbers JP19H04114, JP22H03694 and JP22K19802, JST CREST Grant Number JP-JCR22M2, NEDO Grant Number JPNP20006, and AMED Grant Number JP21zf0127005.

Declarations

Conflict of interest The authors have no competing interest to declare that are relevant to the content of this article.

References

1. Ackermann MR, Märtens M, Raupach C et al (2012) Streamkm++ a clustering algorithm for data streams. J Exp Algorithmics (JEA) 17:2–1
2. Angiulli F, Fassetti F (2007) Detecting distance-based outliers in streams of data. In: Proceedings of the 16th ACM conference on information and knowledge management, pp 811–820
3. Angiulli F, Pizzuti C (2005) Outlier mining in large high-dimensional data sets. IEEE Trans Knowl Data Eng 17(2):203–215
4. AT&T The ORL database of faces. https://cam-orl.co.uk/facedatabase.html
5. Breunig MM, Kriegel HP, Ng RT et al (2000) LOF: identifying density-based local outliers. In: Proc. 2000 ACM SIGMOD international conference on management of data, pp 93–104
6. Cao X (2021) High-dimensional cluster boundary detection using directed Markov tree. Pattern Anal Appl 24(1):35–47

7. Cao F, Estert M, Qian W et al (2006) Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM international conference on data mining. SIAM, pp 328–339
8. Cao L, Yang D, Wang Q et al (2014) Scalable distance-based outlier detection over high-volume data streams. In: 2014 IEEE 30th International conference on data engineering. IEEE, pp 76–87
9. Cao X, Qiu B, Xu G (2019) Bordershift: toward optimal meanshift vector for cluster boundary detection in high-dimensional data. Pattern Anal Appl 22(3):1015–1027
10. Cox L. Bio medical data. http://lib.stat.cmu.edu/datasets/
11. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
12. Ester M, Kriegel HP, Sander J et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp 226–231
13. Gao J, Ji W, Zhang L et al (2020) Cube-based incremental outlier detection for streaming computing. Inf Sci 517:361–376
14. Hariri S, Kind MC, Brunner RJ (2019) Extended isolation forest. IEEE Trans Knowl Data Eng 33(4):1479–1489
15. Hawkins DM (1980) Identification of outliers, vol 11. Springer, Berlin
16. Ishida K, Kitagawa H (2008) Detecting current outliers: Continuous outlier detection over time-series data streams. In: International conference on database and expert systems applications. Springer, Berlin, pp 255–268
17. Jin W, Tung AK, Han J (2001) Mining top-n local outliers in large databases. In: Proc. 7th ACM SIGKDD international conference on knowledge discovery and data mining, pp 293–298
18. Khalique V, Kitagawa H (2021) VOA*: fast angle-based outlier detection over high-dimensional data streams. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, pp 40–52
19. Khalique V, Kitagawa H (2022) BPF: an effective cluster boundary points detection technique. In: Strauss C, Cuzzocrea A, Kotsis G et al (eds) Database and expert systems applications. Springer, Cham, pp 404–416
20. Knorr EM, Ng RT, Tucakov V (2000) Distance-based outliers: algorithms and applications. Very Large Data Bases J 8(3–4):237–253
21. Knox EM, Ng RT (1998) Algorithms for mining distancebased outliers in large datasets. In: Proceedings of the international conference on very large data bases, pp 392–403
22. Kontaki M, Gounaris A, Papadopoulos AN et al (2011) Continuous monitoring of distance-based outliers over data streams. In: 2011 IEEE 27th International conference on data engineering. IEEE, pp 135–146
23. Kriegel HP, Schubert M, Zimek A (2008) Angle-based outlier detection in high-dimensional data. In: Proc. 14th ACM SIGKDD international conference on knowledge discovery and data mining, pp 444–452
24. Kriegel HP, Kröger P, Zimek A (2009) Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans Knowl Discov Data (TKDD) 3(1):1–58
25. LeCun Y, Cortes C. Mnist. http://yann.lecun.com/exdb/mnist/
26. Li Y, Maguire L (2010) Selecting critical patterns based on local geometrical and statistical information. IEEE Trans Pattern Anal Mach Intell 33(6):1189–1201
27. Li X, Wu X, Lv J et al (2018) Automatic detection of boundary points based on local geometrical measures. Soft Comput 22(11):3663–3674
28. Liu FT, Ting KM, Zhou ZH (2008) Isolation forest. In: 2008 eighth IEEE international conference on data mining. IEEE, pp 413–422
29. Na GS, Kim D, Yu H (2018) Dilof: effective and memory efficient local outlier detection in data streams. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp 1993–2002
30. Papadimitriou S, Kitagawa H, Gibbons PB et al (2003) LOCI: fast outlier detection using the local correlation integral. In: Proceedings 19th international conference on data engineering. IEEE, pp 315–326
31. Papadopoulos A, Manolopoulos Y (1997) Performance of nearest neighbor queries in r-trees. In: International conference on database theory. Springer, Berlin, pp 394–408
32. Pham N, Pagh R (2012) A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In: Proc. 18th ACM SIGKDD international conference on knowledge discovery and data mining, pp 877–885
33. Pokrajac D, Lazarevic A, Latecki LJ (2007) Incremental local outlier detection for data streams. In: 2007 IEEE symposium on computational intelligence and data mining. IEEE, pp 504–515
34. Qiu B, Cao X (2016) Clustering boundary detection for high dimensional space based on space inversion and Hopkins statistics. Knowl Based Syst 98:216–225
35. Qiu BZ, Yue F, Shen JY (2007) BRIM: an efficient boundary points detecting algorithm. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, pp 761–768
36. Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data, pp 71–79

37. Salehi M, Leckie C, Bezdek JC et al (2016) Fast memory efficient local outlier detection in data streams. IEEE Trans Knowl Data Eng 28(12):3246–3260
38. Schubert E, Sander J, Ester M et al (2017) DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. ACM Trans Database Syst (TODS) 42(3):1–21
39. Tran L, Fan L, Shahabi C (2016) Distance-based outlier detection in data streams. Proc VLDB Endow 9(12):1089–1100
40. Xia C, Lu H, Ooi BC et al (2004) Gorder: an efficient method for KNN join processing. In: Proc. 30th international conference on very large data bases, vol 30, pp 756–767
41. Xia C, Hsu W, Lee ML et al (2006) Border: efficient computation of boundary points. IEEE Trans Knowl Data Eng 18(3):289–303
42. Yoon S, Lee JG, Lee BS (2019) NETS: extremely fast outlier detection from a data stream via set-based processing. Proc VLDB Endow 12(11):1303–1315
43. Zimek A, Schubert E, Kriegel HP (2012) A survey on unsupervised outlier detection in high-dimensional numerical data. Stat Anal Data Min ASA Data Sci J 5(5):363–387

Vijdan Khalique received his B.Eng. and M.Eng. degrees in Software Engineering from Mehran University of Engineering and Technology (MUET), Pakistan. He served as a Lecturer in the department of Software Engineering MUET, Pakistan. Currently, he is a Ph.D candidate working under the supervision of Prof. Hiroyuki Kitagawa in Knowledge and Data Engineering (KDE) Lab at the University of Tsukuba, Japan. He is a member of Information Processing Society Japan (IPSJ), IEEE Computer Society and Pakistan Engineering Council (PEC). His research interests include data mining and deep neural networks for data analysis.

Hiroyuki Kitagawa received the B.Sc. degree in physics and the M.Sc. and Dr.Sc. degrees in computer science, all from the University of Tokyo. He is currently a full professor at International Institute for Integrative Sleep Medicine, University of Tsukuba. He is also a Collaborative Fellow at Center for Computational Sciences, University of Tsukuba and an Invited Researcher at Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology. His research interests include databases, data mining, stream processing, and sleep data analysis. He is an IEICE Fellow, an IPSJ Fellow, a member of ACM and IEEE, and an Associate Member of the Science Council of Japan.

Toshiyuki Amagasa received B.E., M.E., and Ph.D from the Department of Computer Science, Gunma University in 1994, 1996, and 1999, respectively. He is currently a full professor at the Center for Computational Sciences, University of Tsukuba. His research interests cover database systems, data mining, and database application in scientific domains. He is a senior member of IEICE and IEEE, and a member of DBSJ, IPSJ, and ACM.