World Wide Web (2023) 26:3759–3794
https://doi.org/10.1007/s11280-023-01204-9

Efficient continuous kNN join over dynamic high-dimensional data

Nimish Ukey[1] · Guangjian Zhang[1] · Zhengyi Yang[1] · Binghao Li[2] · Wei Li[3] · Wenjie Zhang[1]

Received: 27 February 2023 / Revised: 22 August 2023 / Accepted: 22 August 2023 /
Published online: 11 September 2023

Abstract
Given a user dataset $\boldsymbol{U}$ and an object dataset $\boldsymbol{I}$, a kNN join query in high-dimensional space returns the $\boldsymbol{k}$ nearest neighbors of each object in dataset $\boldsymbol{U}$ from the object dataset $\boldsymbol{I}$. The kNN join is a basic and necessary operation in many applications, such as databases, data mining, computer vision, multi-media, machine learning, recommendation systems, and many more. In the real world, datasets frequently update dynamically as objects are added or removed. In this paper, we propose novel methods of continuous kNN join over dynamic high-dimensional data. We firstly propose the HDR$^+$ Tree, which supports more efficient insertion, deletion, and batch update. Further observed that the existing methods rely on globally correlated datasets for effective dimensionality reduction, we then propose the HDR Forest. It clusters the dataset and constructs multiple HDR Trees to capture local correlations among the data. As a result, our HDR Forest is able to process non-globally correlated datasets efficiently. Two novel optimisations are applied to the proposed HDR Forest, including the precomputation of the PCA states of data items and pruning-based kNN recomputation during item deletion. For the completeness of the work, we also present the proof of computing distances in reduced dimensions of PCA in HDR Tree. Extensive experiments on real-world datasets show that the proposed methods and optimisations outperform the baseline algorithms of naive RkNN join and HDR Tree.

Keywords K nearest neighbors · KNN join · Dynamic data · High-dimensional data

1 Introduction

The *k*-Nearest Neighbor (kNN) join problem is fundamental in many data analytic and data mining applications, such as classification [1–3], clustering [4, 5], outlier detection [6–10], similarity search [11–13], etc. It can also be applied in some applications of the healthcare

Nimish Ukey and Guangjian Zhang contributed equally to this work.

✉ Zhengyi Yang
zhengyi.yang@unsw.edu.au

Extended author information available on the last page of the article

domain, such as for anomalies detection in healthcare data [14], multiclass classification [15], emotion classification [16], similarity search [17], to detect autism spectrum disorder (ASD) children [18], etc. Given a query dataset U and an object dataset I in high-dimensional space, a kNN join query returns the kNN of *ALL* objects in dataset U from dataset I. For example, social media platforms like YouTube, Netflix, Twitter, Facebook, and others use kNN join to represent people and content as feature vectors in a high-dimensional space so it can make suggestions based on what people like. E-commerce recommendation systems use kNN join similarly to suggest products to customers so that they are more likely to buy them.

In many modern uses of kNN join, like the ones listed above, data is being created at a very fast rate. According to Twitter, approximately 350, 000 in tweets were sent per minute [19]. In many modern uses of kNN join, data is being created at a very fast rate. To utilise the newly generated data to provide an up-to-date and timely response, there emerges a demand for an efficient kNN join on highly dynamic data.

We can see from existing work that the vast majority of existing kNN join approaches [9, 20–24] work with static data. For these methods to work with dynamic data, the kNN join to be recalculated from scratch every time the object dataset is updated, such as when a new object is added, or an old one is removed. This leads to massive processing time and causes extremely high latency. Yu et al. [25] devised the high-dimensional kNNJoin$^+$ algorithm to dynamically update new data points, enabling incremental updates on kNN join results. But because it was a disk-based technique, it could not meet the real-time needs of real-world applications. Further work by Yang et al. [26] proposes the index structure of High-dimensional R-tree (HDR Tree) on dynamic kNN join (DkNNJ). It identifies data nodes whose kNN are affected by the inserted data and updates only the affected data points to avoid redundant computation. In addition, HDR Tree performs dimensionality reduction through principal component analysis (PCA) and clustering to further prune candidates.

For update operations, insertion and deletion are the most fundamental operations. Referring to existing techniques, they primarily focus on the insertion operation. For every deletion of a data item, they have to recompute the kNN for all query points as in static solutions, which results in high time complicity and inefficiency. The results of a kNN join are updated by existing algorithms on every update operation, and none of them supports batch updates. Considering the fast growth of high-velocity streaming data, these approaches significantly limit the performance of dynamic kNN join on large datasets. To address these issues, we came up with lazy updates, batch updates, and optimised deletions in our previous work [27]. We design a lazy update mechanism. It identifies the users whose kNN should be updated on insertions and deletions and marks them as "dirty" nodes in the HDR Tree. The actual updating computation is delayed until the kNN values of the affected users are required. In batch updates, for a given batch of updates (i.e., insertions and deletions), we propose not to update the results immediately for each new item. Instead, we find out which users are affected by the batch of updates before we update them. It helps avoid redundant computation. Item deletions in kNN join are costly operations. We need to search all affected users and update their kNN list for any deletion operation. Thus, we propose to maintain a reverse kNN table for all items to speed up the process of searching for affected users.

This paper extends the paper Efficient kNN Join over Dynamic High-dimensional Data [27]. Compared to the conference version, we further identify and address the following problems in existing solutions for kNN join over dynamic high-dimensional data, which are listed below:

1. *Non-globally correlated data.* Existing algorithms [25–27] heavily rely on global correlation in the datasets for effective dimensionality reduction. However, real-world datasets

are usually not globally correlated [28, 29]. Consequently, existing algorithms may fail to capture distinct features on non-globally correlated data.
2. *Redundant PCA Computation.* Earlier, every time a new item was inserted or deleted, we had to recompute the transformed dimensionality of that item based on the dimension of that level. This creates redundant PCA computation, which is very costly.
3. *Inefficient Recomputation during Deletion.* For each deletion of object data, they need to recompute the kNN for all query points as in static solutions, which results in high time complicity and inefficiency. In our previous work [27], we proposed maintaining the RkNN table to accelerate the process of RKNN search when deletion. However, the updating process of the affected kNN lists is still costly and can be further optimised.

To address the above shortcomings, we present new techniques and optimisations for kNN join over dynamic high-dimensional data in this paper. Our proposed techniques support efficient searching, improvised insertion, and optimised deletion. It computes the kNN join in the main memory and adopts our new HDR Forest to reduce the cost of the in-memory search. Further optimisations are presented in this work to enhance the effectiveness of our approach. Compared to the conference version being extended [27], the additional contributions of this paper can be summarised as follows:

1. *The HDR Forest.* We propose the HDR Forest to provide efficient processing of non-globally correlated datasets. An HDR Forest constructs multiple local HDR Trees built by local data and local PCA matrices. It fully makes use of the PCA features of local correlation instead of global correlation to capture distinct features among non-globally correlated datasets.
2. *Precomputation of the PCA States.* We propose to precompute the PCA states of all item vectors in advance and index them in the tree. By doing so, we managed to avoid the redundant PCA computation in previous algorithms to speed up updates.
3. *Pruning-based kNN Recomputation during Deletion.* We propose a pruning-based optimisation during deletion. Specifically, we derive an upper bound of distance to prune unpromising items when recomputing the kNN of affected items. This optimisation significantly reduces the number of computing the distance on full-dimensions between the affected user and all the items in the sliding window when items are deleted.
4. *Extensive Experiments and Additional Proof.* Extensive experimental analysis on various real-world high-dimensional datasets has been conducted to evaluate the performance of the proposed methods. The results show that our HDR$^+$ Tree and HDR Forest significantly outperform the existing algorithms of continuous kNN join over dynamic high-dimensional data. In addition, we present the proof of the theorem that $dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2)$ used in HDR Tree for the completeness of the paper, as the previous report [30] with the proof is no longer accessible.

Outline The rest of this paper is organised as follows. Initially, we define the problem and discuss preliminaries in Section 2. We review the related work in Section 3. We dedicate Section 4 for a brief summarisation of the HDR Tree, and we also provided the proof of distance computation in the reduced dimension theorem. In Section 5, we discussed the previous optimisations, followed by the proposed ones. We discuss the proposed HDR Forest in Section 6. Section 7 presents the experimental results and analysis, followed by a conclusion in Section 8.

2 Background

2.1 Problem definition

This section provides the definitions for kNN join, reverse kNN join, kNN search, and dynamic kNN join operations. Throughout this paper, we refer to U and I as the *User* and *Item* sets for kNN join operations. Table 1 summarises frequently used symbols. We focus on the problem of continuously processing a kNN join while handling a *sliding window* of items. For each user in $U = \{u_1, u_2, \ldots, u_n\}$ we search for the k nearest neighbor items in the item set $I = \{i_1, i_2, i_3, \ldots, i_k\}$ in d-dimensional space. We formally define kNN join in Definition 1.

Definition 1 (kNN Join) Let $U = \{u_1, u_2, \ldots, u_n\}$ be a user dataset of a set of user data points, and $I = \{i_1, i_2, \ldots, i_m\}$ be an item dataset of a set of item data points in d-dimensional space $\mathbb{R}^d$, the function $d(u_i, i_j)$ to compute the distance between two data points u_i and i_j be the Euclidean distance function, and k be a positive natural number. Then, the result of the kNN Join query is a set $kNNJ(U, I, k) \subseteq U \times I$, which includes for every point of

Table 1 Summary of symbol and definitions

Symbols	Definitions
U, I	User set and Item set
u, i	User and Item
W	Sliding window
k	Number of nearest neighbors
d, r	Dimensionality of original and reduced dataset
R, Rp	kNN and RkNN set
f	Fanout
l	Level of tree
$d(l)$	Number of dimensions reserved on the l level of the HDR Tree
$PCA(i)$	The low-dimensioal vector produced by PCA projection from the item i
$PCA_d(i)$	The vector with d dimensions reserved produced by PCA from the item i
L	Height of tree
C_i	The i-th Cluster
$dist$	Distance
$dist_{PCA}$	Distance of two low-dimensional vectors produced by PCA transformation
$dknn$	Distance from the user to its k-th nearest neighbor
$dist_{PCA(d)}$	Distance of two low-dimensional vectors produced by the first d row vectors of the PCA matrix from the full-dimensional vectors
$maxdknn$	Maximal $dknn$ value of the users in a cluster
X, V, T	Matrix
n, m, N, M	The size of a set
$radius$	The radius of a cluster on full-dimensions
$radius_{PCA}$	The radius of a low-dimensional cluster whose user vectors are produced by PCA transformation from their full-dimensional states.
$center$	The center of a cluster

$U(u_i \in U)$, it finds the k closest neighbours in I : $kNNJ(U, I, k) = \{(u_i, i_j) : u_i \in U, i_j \in kNN(I, u_i, k)\}$.

For any update operation in dynamic data, it is necessary to search for the affected users [26]. This refers to a *reverse kNN (RkNN) join* as defined in Definition 2. We use RkNN [31] to check for the set of users affected by an item's insertion/deletion operation and update the kNN result accordingly.

Definition 2 (Reverse kNN Join) $U = \{u_1, u_2, \ldots, u_n\}$ be a user dataset of a set of user data points, and $I = \{i_1, i_2, \ldots, i_m\}$ be an item dataset of a set of item data points in d-dimensional space $\mathbb{R}^d$, the Euclidean distance function $d(u_i, i_j)$ compute the distance between two data points u_i and i_j and the natural number $k \in \mathbb{N}^+$. Then, the results of reverse kNN Join with respect to the query data point i_j is a set of data points $RkNN(U, i_j) \subseteq U$ that includes i_j as one of their kNNs. $RkNN(U, i_j, k) = \{u_1, u_2, \ldots, u_n\} \subseteq U$, such that $\forall u \in U \wedge i_j \in I$.

Definition 3 (kNN Search) Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of user data points and $I = \{i_1, i_2, \ldots, i_m\}$ be a set of item data points in d-dimensional space $\mathbb{R}^d$, u_q be a user query point in $\mathbb{R}^d$, the function $d(u_q, i_i)$ to compute the distance between two data points u_q, and i_i be the distance function and k be a positive natural number. Then, the result of the kNN Search with respect to u_q and I is an ordered collection, $kNN(I, u_q, k) \subseteq I$, which contains $k(1 \leq k \leq |I|)$ different data points with the k least distances for u_q, such that $kNN(I, u_q, k) = \{i_1, i_2, \ldots, i_k\} \subseteq I$, $d(u_q, i_i) \leq d(i_j, u_q)$ if $1 \leq i < j \leq k$ and $\forall i \in I \mid kNN(I, u_q, k)$; we have $d(u_q, i_i) \leq d(i, u_q)$, $1 \leq i \leq k$.

Definition 4 (Dynamic kNN Join) Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of user data points and $I = \{i_1, i_2, \ldots, i_m\}$ be a set of item data points in d-dimensional space $\mathbb{R}^d$, u_q be a user query point in $\mathbb{R}^d$, the Euclidean function $d(u_q, i_j)$ to compute the distance between two data points u_q, and i_j be the distance function and k be a positive natural number. and k be a positive natural number. Then, the dynamic kNN join is the ability to dynamically join the similar data points $DkNNJ(U, I, k) \subseteq U \times I$ in $\mathbb{R}^d$, which includes for every point of U its $k(1 \leq k \leq |I|)$ closest neighbours in I : $DkNNJ(U, I, k) = (u_i, i_j) : \forall u_i \in U, i_j \in kNN(I, u_i, k)$ and maintains (updates) the complete join result with every update operation, i.e., for insertion or deletion of any data item $i_j \in I$, finding the affected user set $u_a : RkNN(i_j) \mid i_j \in I \wedge RkNN(i_j) \subset U$ and updating the affected user set $kNN(I, u_a, k) \subseteq U \times I$.

We denote kNN join over dynamic data as *Dynamic kNN Join (DkNNJ)*. DkNNJ maintains a recommendation list R_j for every user u_i, containing its k nearest neighbors. We also used the *sliding window* to monitor the item stream, and only the items within the sliding window are recommended. The sliding window must also be updated every time an item is updated. According to the pruning strategy, any newly inserted item will affect the set of users if and only if it falls within the user's *dknn* range.

If any existing item is removed or expired, then we have to check for the set of affected users. Accordingly, we update the recommendation list of those affected users. Similarly, when any new item is inserted into the sliding window, it looks for the affected user and updates its recommendation list. It must be updated dynamically in a real-time system.

Lemma 1 *The user is affected by the newly inserted item i_j or the expired item i_j if and only if it falls within the user's dknn radius range.*

The lemma states that the purpose is to create an index on the user dataset U using the *dknn* value, which can subsequently be used to look up the impacted users for every updated

(i.e., new or expired) data item. We have to update the recommendation list R_j of every affected user whose item i_j has been updated. Considering i_j as a new data item, we add i_j into R_j and order R_j to maintain the kNN results. If an item i_j has expired, it is removed from R_j. When an item i_j expires, we must recompute R_j because R_j will contain fewer than k results.

Problem definition Given a user dataset U and an item dataset I, our goal is to dynamically output the kNN join results of U in I upon every update of I (i.e., insertions and deletions).

3 Related work

Static kNN join The dimensionality reduction is the transformation of the high-dimensional dataset into a low-dimensional dataset while keeping some useful properties of the original dataset. It is a common approach to improve the performance of the kNN for the higher-dimensional dataset. The Pyramid Technique [32] iDistance [33, 34], Δ-tree [35], VA-File [36], iMinMax [37, 38], LSH [39], LSHI [40]; and other approaches have been proposed to deal with the curse of dimensionality. It usually helps to improve the performance of the kNN for high-dimensional datasets.

In the literature, many studies have been carried out on index structures that can handle kNN join. In reference [20, 41], researchers proposed the very first "kNN join" work, where they came up with a new method called "multi-page indexing (MuX)" to calculate the kNN join. It utilises the index-nested loop join approach and follows the R-tree [42] structure. Thus, large-size pages (hosting pages) were employed to reduce the I/O time. It also uses buckets, which are smaller minimum bounding rectangles, to split the data more accurately and efficiently. Every set of objects (i.e., R and S) has its own index, and MuX iterates the index pages on R. Pages of S are retrieved using the index and a search for the k nearest neighbors of pages of R stored in the main memory. The procedure comes to an end after all pages have been looked through or filtered out. Combining bucket selection and page loading techniques improves MuX's performance. However, this approach has some limitations, such as performance degradation as dimensionality increases and high memory overhead, which limits the scalability of MuX kNN Join.

To address the limitations of the MuX technique, Xia et al. proposed a new kNN join algorithm called Gorder (G-ordering) [21], a block-nested loop join approach. It uses sorting, join scheduling, and filtering based on the distance to keep I/O and CPU costs as low as possible. The input datasets are sorted into G-order before the scheduled block nested loop join is applied to the G-ordered data. Gorder's joining stage is defined by two characteristics. First, it optimises I/O and CPU times individually using a two-tier partitioning technique. It then schedules the data joining to speed up the kNN processing. Due to a large number of dimensions in the data, it is important to do as few distance calculations as possible to maximise CPU efficiency. As a result, they created an algorithm to reduce the distance calculation by pruning the unnecessary distance calculations. It manages high-dimensional data well while being simple and effective, but it requires significant computation cost [43].

In [22], the authors propose a novel index-based kNN join technique that makes use of the iDistance as the underlying index structure. In this work, they look at the problem of processing a kNN join and also address the problem of the MuX technique. They devised three strategies: the basic strategy, which they call iJoin, and two improved versions, called iJoinAC and iJoinDR. In the first enhanced version (iJoinAC), approximation bounding cubes were

used to reduce the amount of kNN computation and disk use. Later developments minimised I/O and CPU costs by using the reduced dimensions of the data space (iJoinDR).

To address the computational challenges of large-scale kNN join for data in data mining, Jiaqi et al. [44] propose a Spark-based approach that employs Locality-Sensitive Hashing to group similar data points in buckets, effectively reducing candidates for kNN. This technique showcases accuracy and efficiency, especially for high-dimensional big data with varying dimensions.

The work in [45] fulfils the gap of adding result diversification to kNN joins by means of Influence criteria which is a coverage-based criterion defined by the distance-based ternary relationship between each search reference, the dataset, and the result set in large-scale, big data frameworks. This paper solves the problem by extending the nested Better Results with Influence Diversification algorithm to a Map-Reduce framework.

But since these techniques [20–22, 44, 45] use a static dataset, when the dataset is updated, every user must go through the time-consuming and expensive kNN calculation.

Dynamic kNN join In [46], authors focus on efficient computation of continuous kNN joins in high-dimensional data for real-time social recommendations. They introduce a binary sketch-based method that uses Hamming distances to reduce computational load. This technique improves performance in updating kNN join results as new data enters or exits a sliding time window.

In [40], researchers address the challenge of real-time recommendation in social networking platforms by kNN Join on high-dimensional data using an approach called Locality-Sensitive Hashing-based Index (LSHI) that focuses on the user sets to efficiently identify users affected by new data updates. Through experiments, the paper demonstrates the effectiveness of LSHI in maintaining real-time kNN recommendation results.

This research addresses the challenge of performing efficient kNN join over dynamic data streams in location-aware systems. They introduce an adaptive scalable stream kNN join technique [47], named ADS-kNN, which optimises kNN queries for distributed environments by using a multistage execution plan that overlaps computation and communication stages. Experimental results on a 56-core system demonstrate that ADS-kNN achieves significantly higher throughput than single-threaded and round-robin partitioning approaches.

In this work [48], Lee et al. address efficient kNN join query processing for large-scale data using MapReduce. Their approach introduces seed-based dynamic partitioning to reduce index construction overhead. By utilising average distances between seeds, computational load for candidate partition selection is minimised. Performance analysis demonstrates improved query processing time compared to existing methods.

Also, a novel and helpful variation of the Reverse Nearest Neighbor query, known as the Reverse Nearest Neighbourhood (RNNH) query [49], has been discovered. This query aims to identify the set of neighboring facilities for which the queried point serves as the closest among all available facilities. Considering its applicability, it can be considered for further research. The recent research works, i.e. "approximate kNN Join" [40, 46] and "Distributed kNN Join" [47, 48], concentrate on methods that are not centralised or exact solutions. In contrast, our study exclusively emphasises the exact centralized approach to kNN join.

Yu et al. proposed the kNNJoin$^+$ technique [25] for processing kNN join queries on high-dimensional data. Four different types of data structures were used in this study: the RkNN join table, kNN join table, iDistance, and the sphere tree. The M-tree [50] concept is used to create the sphere tree. It's structured similarly to an R-tree, but it deals with spheres rather than rectangles. The iDistance indexing is used to find the kNN for a newly inserted point p while, on the other hand, the sphere tree is used to look for RkNN, i.e., points with p

as their kNN. The iDistance approach used the Pyramid Technique [32] to convert a high-dimensional space into a one-dimensional value. In this work, they develop a shared query optimisation strategy in order to improve performance, but being a disk-based approach, it is difficult to meet real-time requirements.

Yang et al. [26] provide two different data structures in a continuous kNN join namely exact (HDR Tree) and approximate (HDR* Tree) solutions. HDR Tree utilises the PCA [51] and clustering approaches for dimensionality reduction. On the other hand, HDR* Tree employs the Random Projection [52] method for approximate dynamic kNN join. It uses a random matrix to translate the data from d to r dimensions.

4 HDR tree

To handle dynamic kNN join in high-dimensional data, we adopt the HDR Tree [26] as the index structure in our method. HDR Tree performs dimensionality reduction via PCA and clustering. In this section, we briefly introduce HDR Tree.

4.1 Principal component analysis

The PCA approach is basically used to reduce the cost of computation in the tree structure. Consider the dataset $X_{N\times d}$, where N represents the number of rows and d represents the dataset's dimensionality. We must follow a specific procedure to convert the dataset from d to r dimensions. The value of r is $0 < r < d$. While processing the dimensionality reduction, the direction having the highest variance is considered the first principal component and is followed by the second component in descending order. The tree structure's 1^{st} dimension consists of values with high variance.

1. Initially, the covariance matrix of the input dataset (X) is computed.
2. Using the covariance matrix, eigenvalues and their corresponding eigenvectors are calculated as follows:
$$COV = \frac{\sum_{i=1}^{n}(X_i - \overline{x})(Y_i - \overline{y})}{n-1}$$
3. Sort the eigenvalues and associated eigenvectors in descending order.
4. The transformed matrix (T) is used to transform the newly added point to the necessary dimension.

This different dimensionality approach provides better pruning power, which helps reduce the computation overhead.

4.2 Structure of tree

Figure 1 illustrates the structure of the HDR Tree. At each level, we utilise different dimensions with the help of eigenvalues. The high-dimensional dataset was partitioned into clusters using the k-means clustering method. Other clustering approaches can also be utilised. The root level (i.e., level 1) with d_1 dimensionality is clustered (i.e., $C_1, C_2, \ldots, C_f$) based on the fanout value f. The dimensionality d_2 is set to the following level (level 2). Every cluster from the root node is sub-clustered. In our case, cluster C_1 is subdivided into $C_{11}, C_{12}, \ldots C_{1f}$.The dimensionality increases at every level. At level l, the leaf node has its full dimensionality, d_l.

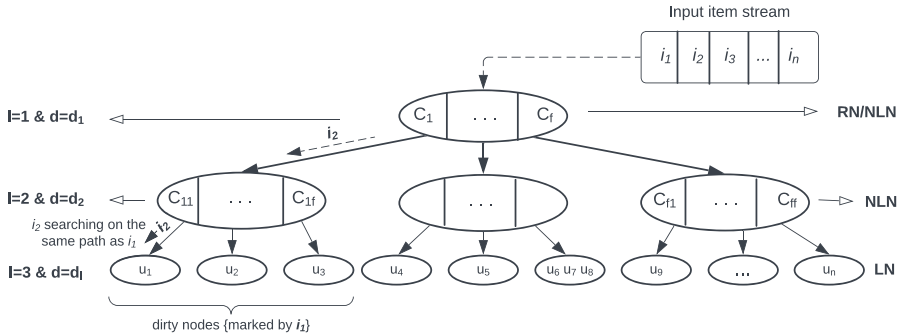

Figure 1 Example of lazy updates on HDR Tree

In Figure 1, the internal nodes and the root node have the same structure, represented by a tuple $(C, maxdknn, l, num, r, ptr)$. Where C is the center of the sub-cluster, $maxdknn$ is the maximum distance between the users and its kNN, l is the level of the node, num is considered the number of users, r is the radius of the sub-cluster, and ptr is the pointer to the next node.

4.3 Construction

The estimated height of the tree is calculated using $L = [\log_f N]$ to build the HDR Tree. The fixed fanout size is used, and the threshold value θ is adjusted so that the tree's level does not exceed L too much. When the hierarchical level of a cluster during the construction process surpasses L, and the cluster's size exceeds θ, we persist in partitioning the cluster using the entire user set's full dimensionality, implying $d_l = d$ (when $l > L$). For each level, the dimensionality is calculated and stored for further processing. First, we create the cluster with d_1 dimensions and set the $maxdknn$ value. The maximum distance between the cluster's users and their k-th nearest neighbor item is defined as $maxdknn$.While constructing a tree, it checks for the number of users within a cluster. If it is less than a θ, it creates the leaf node (LN); otherwise, it creates the non-leaf node (NLN) using the new incremental dimensionality approach.

4.4 Search

When the search algorithm looks for the affected users in a leaf node LN, it directly computes the distance between the users in LN and the updated item. Contrary to this, for a cluster in a non-leaf node, the pruning condition will be checked to decide if the subtree of that cluster will be searched or pruned. For a cluster C_j in a non-leaf node, the pruning condition $dist_{PCA}(i, center(C_j)) - radius_{PCA}(C_j) \leq maxdknn$ will be checked against an updated item i. (Note that the applied pruning condition on the HDR Tree for RKNN search is explained in Theorem 1). If the pruning condition is satisfied, the subtree of C_j needs not to be visited further because the users in it will not be affected by the updating of i. Otherwise, the search should be done in the C_j subtree. The Theorem 1 is defined further below.

Theorem 1 (Pruning on a HDR Tree) *When a cluster C_j in an HDR Tree is visited during the RkNN search for an updated item i, if $dist_{PCA}(i, center(C_j)) - radius_{PCA}(C_j) >$*

$maxdknn(C_j)$, the C_j subtree can be pruned, and we can confirm that all the users in C_j may not be affected by i. Otherwise, the subtree of C_i should be searched.

Proof When $dist_{PCA}(i, center(C_j)) > radius_{PCA}(C_j) + maxdknn(C_j)$, we can have:

$$dist(u, i) \geq dist_{PCA}(u, i) \tag{1}$$

The reason of the inequality 1 is the Theorem 2 that $dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2)$. The proof for Theorem 2 is in Subsection 4.5. Then we can get that:

$$dist_{PCA}(u, i) \geq dist_{PCA}(center(C_i), i) - dist_{PCA}(center(C_i), u) \tag{2}$$

The inequality 2 is based on the *Triangle Principle* that $dist(a, b) - dist(b, c) \leq dist(a, c)$ in Euclidean distance space where $dist(a, b) = (a^2 + b^2)^{\frac{1}{2}}$. We have all of our user and item vectors in Euclidean distance space. Then the inequality below can be gained:

$$dist_{PCA}(u, i) \geq dist_{PCA}(center(C_i), i) - radius_{PCA}(C_i) \tag{3}$$

Note that the radius of a cluster is the longest distance between its center and its member vectors. We can deduce from the precondition, inequality 3, and inequality 1 that

$$dist(u, i) \geq dist_{PCA}(u, i) > maxdknn(C_i) \geq maxdknn(u) \tag{4}$$

The above analysis shows that when a $dist_{PCA}(i, center(C_j)) - radius_{PCA}(C_j) > maxdknn(C_j)$, the users in C_j are unlikely to be affected by updating i, so C_j can be pruned.

□

Theorem 2 ($Dist_{PCA}$ is a low bound of $dist$) *The distance between any two data points in PCA space is always no larger than the distance between them in the full-dimensional space, i.e., $dist(T_0 \cdot V_1, T_0 \cdot V_2) \leq dist(V_1, V_2)(dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2))$, where T_0 is a partial PCA matrix, also known as Projection Matrix. Note that arbitrary two data points are available, no matter if they are from the dataset that produces the PCA matrix or not.*

4.5 Proof of PCA lower bound

Theorem 2 states that a set of vectors named N with n dimensions must have a PCA matrix T whose first r (r is an arbitrary number between 1 and n) or entire rows can form a partial matrix T_0 of T to allow the inequality $dist(T_0 \cdot V_1, T_0 \cdot V_2) \leq dist(V_1, V_2)(dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2))$ to hold, where V_1 and V_2 are two arbitrary n-dimensional vectors. This is the theoretical basis for the pruning condition, which states that if $dist_{PCA}(i, center(C_j)) > maxdknn(C_j)$, then $dist(i, u) > dknn(u)$ must hold for an arbitrary user vector u in cluster C_j.

Note that Theorem 2 is widely used in the literature [26, 27, 29, 35, 51, 53] where PCA is used to reverse the spatial features of high-dimensional data. All these papers refer to a report [30] and state that the proof of Theorem 2 is in the report. However, the link to such a report is no longer accessible. For the completeness of this paper, we present the proof of Theorem 2 as follows.

Proof For the given vector set named N with n dimensions, try to find a PCA matrix T of it whose first r (r is an arbitrary number between 1 and n.) or whole rows can be composed into a partial matrix T_0 of T to let $dist(T_0 \cdot V_1, T_0 \cdot V_2) \leq dist(V_1, V_2)(dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2))$ hold. If such T can be found, the theorem is proved.

Step 1. Calculate the covariance matrix M of N, then compute the eigenvalues of M (assuming p eigenvalues), and sort the eigenvalues in descending order into a list $E_0 = \{e_1, e_2, \ldots, e_p\}$.

Step 2. Assume that $e_i (i = 1, 2, \ldots, p)$ has a multiplicity of n_i. Because M is a real symmetric matrix, as a result, $\sum_{i=1}^{p} n_i = n$, and e_i has n_i corresponding eigen vectors $E_{i,1}, E_{i,2}, \ldots, E_{i,n_i}$ that are linear independent. Calculate the corresponding n_i linear independent eigenvectors for each $e_i (i = 1, 2, \ldots, p)$ and combine the obtained eigenvectors into a matrix M_1 by listing them in descending order of their corresponding eigenvalues.

$$M_1 = \left\{E_{1,1} \ldots E_{1,n_1} \ldots E_{i,1} \ldots E_{i,n_i} \ldots E_{p,1} \ldots E_{p,n_p}\right\}$$

Step 3. For $j = 1, 2, \ldots, p$, perform Schmidt orthogonalization [54] on $E_{j,1}, E_{j,2}, \ldots, E_{j,n_j}$ to obtain the new standard orthogonal vector group $F_{j,1}, F_{j,2}, \ldots, F_{j,n_j}$, and add the newly acquired standard orthogonal vector groups to form a new matrix M_2.

$$M_1 = \left\{E_{1,1} \ldots E_{1,n_1} \ldots E_{i,1} \ldots E_{i,n_i} \ldots E_{p,1} \ldots E_{p,n_p}\right\} \Rightarrow M_2 = \left\{F_{1,1} \ldots F_{1,n_1} \ldots F_{i,1} \ldots F_{i,n_i} \ldots F_{p,1} \ldots F_{p,n_p}\right\}$$

To prove M_2^T is a PCA matrix of N:

Pick an arbitrary group of rows $F_{i,1}^T, F_{i,2}^T, \ldots, F_{i,n_i}^T$ from M_2^T $(1 \le i \le p)$ and take an arbitrary row $F_{i,j}^T$ from the group. $F_{i,j}$ is produced by Schmidt orthogonalization based on $E_{i,1}, E_{i,2}, \ldots, E_{i,n_i}$ which is a group of linear-independent eigen vectors of M corresponding to the eigen value e_i, as a result, $|F_{i,j}| = 1 \neq 0$, and $F_{i,j}$ is a linear combination of $E_{i,1}, E_{i,2}, \ldots, E_{i,n_i}$. Therefore, $F_{i,j}$ is an eigen vector of M corresponding to the eigen value e_i. Because the vector group $F_{i,1}, F_{i,2}, \ldots, F_{i,n_i}$ is produced by Schmidt orthogonalization based on $E_{i,1}, E_{i,2}, \ldots, E_{i,n_i}$ which is a group of linear-independent eigen vectors, as a result, $|F_{i,1}| = |F_{i,2}| = \ldots = |F_{i,n_i}| = 1 \neq 0$ and $F_{i,x} \cdot F_{i,y} = 0$ $(x \neq y)$. Therefore, $F_{i,1}, F_{i,2}, \ldots, F_{i,n_i}$ are linear independent. Based on the deduction above, M_2^T is a PCA matrix of N.

To prove $dist(M_0 \cdot V_1, M_0 \cdot V_2) \le dist(V_1, V_2)(dist_{PCA}(V_1, V_2) \le dist(V_1, V_2))$, where M_0 is a matrix composed by the first R (R is an arbitrary number between 1 and n) or the whole rows of M_2^T and V_1 and V_2 are both of n-dimensional:

For two arbitrary vectors $F_{i,x}$ and $F_{j,y}$ $(x \neq y)$, if $i \neq j$, $F_{i,x} \cdot F_{j,y} = 0$ because $F_{i,x}$ and $F_{j,y}$ are two eigen vectors corresponding to two different eigen values (e_i and e_j) of a real symmetric matrix M. Otherwise, $F_{i,x} \cdot F_{j,y} = 0$ because $F_{i,x}$ and $F_{j,y}$ are produced by Schmidt orthogonalization based on the same linear-independent vector group. And $|F_{1,1}| = \cdots = |F_{p,n_p}| = 1$. As a result, M_2^T is a matrix consisting of a standard orthometric base. Therefore, $dist(M_2^T \cdot V_1, M_2^T \cdot V_2) = ((M_2^T \cdot V_1)^T \cdot (M_2^T \cdot V_2))^{\frac{1}{2}} = (V_1^T \cdot M_2^T \cdot M_2 \cdot V_2)^{\frac{1}{2}} = (V_1^T \cdot E \cdot V_2)^{\frac{1}{2}} = (V_1^T \cdot V_2)^{\frac{1}{2}} = dist(V_1, V_2)$. Assuming that $(M_2^T \cdot V_1)^T = (a_1, a_2, \ldots, a_n)$ and $(M_2^T \cdot V_2)^T = (b_1, b_2, \ldots, b_n)$, and M_0 is composed by the first R rows of M_2^T, it will be held that $dist(M_0 \cdot V_1, M_0 \cdot V_2) = (\sum_{i=1}^{R} (a_i - b_i)^2)^{\frac{1}{2}} \le dist(M_2 \cdot V_1, M_2 \cdot V_2) = (\sum_{i=1}^{n} (a_i - b_i)^2)^{\frac{1}{2}} = dist(V_1, V_2)$.

As a result, we can find M_2^T, which is a PCA matrix of N whose first R (R is an arbitrary number between 1 and n) or whole rows can compose a partial matrix M_0 of M_2 to let $dist(M_0 \cdot$

$V_1, M_0 \cdot V_2) \leq dist(V_1, V_2)(dist_{PCA}(V_1, V_2) \leq dist(V_1, V_2))hold$. So the theorem is proved.

□

5 HDR$^+$ tree

In this section, we first discuss our earlier work [27], where we introduce the HDR$^+$ Tree. We proposed lazy updates, batch updates, and deletion optimisation of maintaining the RkNN table in HDR$^+$ Tree.

5.1 Lazy updates

We identify the users whose kNN should be updated for every new item updates. However, we do not update the HDR Tree immediately. Instead, we mark the nodes along the search tree as "dirty," meaning that the radius information on these nodes is not tight. We only update them when another new item accesses the same search path.

Example 1 If $I = \{i_1, i_2, \ldots, i_n\}$ is set of data items and $U = \{u_1, u_2, .., u_m\}$ is a user data set. So, we will consider the users affected by update operations i_1, i_2, and i_3 are: $i_1 = \{u_1, u_2, u_3\}, i_2 = \{u_1, u_5, u_9, u_{16}\}, i_3 = \{u_3, u_7, u_{19}\}$. We look for users who are affected by the update operation, i.e., u_1, u_2, and u_3, for the newly inserted item i_1. We mark all the newly affected user nodes as dirty nodes. It will check for the other affected users during the next insertion of an item i_2. In this case, item i_2 affects the u_1, u_5, u_9, and u_{16}. As we can see, it is searching on the same path because it also consists of the u_1. Hence, we will update it as it tries to access the same path. Basically, we are updating the node if and only if there is a necessity to do so.

As shown in Algorithm 1, lines 1-4 look for the affected users within the leaf node. Lines 5 to 10 examine the node's status, i.e., whether it was previously marked as dirty. If the node was marked as dirty, then we update the node and also need to update the HDR Tree. On the other hand, lines 12-15 deal with the non-leaf node. Basically, it works on the pruning approach of the HDR Tree. The cluster is pruned if the distance is less than the $maxdknn$ value, which helps improve an algorithm's efficiency.

5.2 Batch updates

The HDR Tree index structure efficiently searches the affected users R_p in response to any update operation. Subsequently, the recommendation lists for these affected users need to be updated. However, the sequential method employed for these updates has proven computationally intensive. To address this concern, we propose using a batch approach to reduce the computational costs.

This novel approach first identifies the users affected by newly inserted items. Importantly, we refrain from immediately updating the search path for each new item individually. Instead, updates for multiple new items are collectively processed, i.e. we process all updates at the node left (i.e. leaf node) before updating the parameters on the internal level (i.e. non-leaf nodes) to reduce computational costs.

Algorithm 1 LazySearch.

Ensure: node $node$, item i
Require: affected user set R_p and Lazy nodes
1: **if** $node$ is a LN **then**
2: **for** each users u_j in LN **do**
3: **if** $dist(i, u_j) \leq u_j \rightarrow dknn$ **then**
4: add u_j in R_p;
5: **if** $u_j \rightarrow dirty == true$ **then**
6: **if** $u_j \rightarrow R.size() < k$ **then**
7: $computekNN(W)$;
8: **else**
9: $u_j \rightarrow computeDirtykNN(I)$;
10: $u_j \rightarrow dirty = false$;
11: **end if**
12: **end if**
13: **end if**
14: **end for**
15: **else**
16: $C_p = NLN \rightarrow clusters$;
17: **for** each clusters $C_j : C_p$ **do**
18: **if** $dist_{node.dimension}(i, Cj) < Cj \rightarrow maxdknn$ **then**
19: LazySearch $(NLN \rightarrow children[j], i, m)$;
20: **end if**
21: **end for**
22: **end if**

Example 2 For instance, $I = \{i_1, i_2, \ldots i_n\}$ is a stream of new items. As an instance, when item i_1 influences users u_1, u_8, and u_{11}, and subsequent item i_2 affects users u_8, u_{11}, u_{18}, and u_{25}, updating the search path immediately after every new item is computationally expensive. However, adopting the batch approach yields substantial benefits. In this example, items i_1 and i_2 affect common users, namely u_8 and u_{11}. So, rather than processing the updates separately, we go with batch updates, which avoids individual, costly computation updates and helps to improve efficiency.

The batch update operation is performed on all affected nodes using algorithm 2. Here, R_{p-ins} denotes the set of all affected users. For each individual user within the set of affected users, the algorithm calculates the kNN to update the recommendation list, the $dknn$ value of the cluster, and to further update the $maxdknn$ value. The AdjustMaxDkNN method is used to update the HDR Tree structure.

In the batch update deletion algorithm, the R_{p-del} is a set of affected nodes by deletion operation. In lines 1 to 3 of Algorithm 3, check for the affected users, perform an update operation by computing its kNN, and finally adjust the $maxdknn$ value of the HDR Tree. The computekNN method finds the k closest neighbor for the user vector using a sliding window items stream. It first calculates the distance between the user and the item vector, then finds the k nearest neighbors for the desired user and sets the user's $dknn$ value based on that distance. It's worth noting that deletion operations are inherently more costly compared to insertion, thus necessitating optimisation to enhance performance. For the same reason, we introduce lazy updates and batch optimisation for insertion and deletion operations.

Algorithm 2 Batch update for insertion operation.

Require: Affected user set R_{p-ins}
Ensure: Updated R, $DkNN$ and $maxDkNN$
1: **for** each u_j in R_{p-ins} **do**
2: **for** each i in $u_j \rightarrow R$ **do**
3: $D \leftarrow dist(i, u_j)$;
4: $V \leftarrow tuple(D, i)$
5: **end for**
6: $computekNN(V)$;
7: $u_j \rightarrow dknn = V[k]$;
8: **while** $u_j \rightarrow R.size() > k$ **do**
9: pop u_j from R;
10: **end while**
11: **end for**
12: $AdjustMaxDkNN()$;

Algorithm 3 Batch update for deletion operation.

Require: Affected user set R_{p-del}
Ensure: Updated R, $dknn$ and $maxdknn$
1: **for** each u_j in R_{p-del} **do**
2: **if** $uj \rightarrow R.size() < k$ **then**
3: $computekNN(V)$;
4: **end if**
5: **end for**
6: $AdjustMaxDkNN()$;

Algorithm 4 Update RkNN table.

Require: The updating of an item i
Ensure: The updated $RkNN$ table
1: **if** updating=$Deletion$ **then**
2: **for** each user u_j in $RkNN(i)$ **do**
3: Update $kNN(u_j)$
4: Add u_j to $RkNN$(the k-th nearest neighbor of u_j)
5: **end for**
6: Cancel $RkNN(i)$ from the $RkNN$ table
7: **else**
8: Gain $RkNN(i)$ by the HDR Tree
9: Add $RkNN(i)$ to the $RkNN$ table
10: **for** each user u_j in $RkNN(i)$ **do**
11: Cancel u_j from $RkNN$(the k-th nearest neighbor of u_j)
12: Update $kNN(u_j)$
13: **end for**
14: **end if**

5.3 Maintaining RkNN table for deletions

If an item is deleted, the RkNN join of an item on the HDR Tree needs to be computed again without the help of any other data structures. The computation of the RkNN join for the delete operation is a very costly operation. As a result, we use a structure known as an RkNN table [25], which continuously maps the items in the sliding window to their RkNN lists to reduce computation time. Using an RkNN table helps us to directly get the RkNN list of a deleted item without repeating the RkNN join process on the HDR Tree.

An RkNN table contains rows of data, where the number of rows is equal to the size of the sliding window. To map every item to its RkNN list, each row in an RkNN table contains

the index of an item and the RkNN list of that item. An RkNN list of an item means a list that contains the indexes of all the users that have that item in their kNN lists. The Update RkNN table algorithm is described in detail in Algorithm 4.

6 The HDR Forest

In this section, we introduce the HDR Forest with two additional optimisations. We first describe the motivation of the HDR Forest.

Dependence on global correlation HDR Tree [25–27] heavily relies on the global correlation of the datasets for effective dimensionality reduction. However, in the real world, strong global data correlation is not common. When a single HDR Tree is built on a dataset that is not typically globally correlated, the possibility of the Type I Error that rejects to prune the unaffected users will be large on the low levels of the tree. That is because the number of dimensions corresponding to the low levels of an HDR Tree is so small that the difference between the applied pruning condition and the full-dimensional pruning condition is not trivial. Note that the applied pruning condition is that if $dist_{PCA}(i, center(C_j)) - radius_{PCA}(C_j) > maxdknn(C_j)$, then the subtree of C_j can be pruned while a cluster C_j on an HDR Tree is visited during the RkNN search of an updated item i. The applied pruning condition is stated in Theorem 1. The detailed meaning of "full-dimensional pruning condition" and the difference between them is explained below.

Theorem 3 (Full-dimensional pruning on a HDR Tree) *If* $dist(i, center(C_j)) - radius(C_j) > maxdknn(C_j)$*, the subtree of* C_j *can be pruned when a cluster* C_j *is visited during RkNN search of an updated item* i*. Otherwise, the subtree of* C_j *should be searched.*

Proof When $dist(i, center(C_j)) - radius(C_j) > maxdknn(C_j)$, we can have:

$$dist(u, i) \geq dist(center(C_i), i) - dist(center(C_i), u) \tag{5}$$

The inequality (5) is based on the *Triangle Principle*. Then the inequality below can be gained:

$$dist(u, i) \geq dist(center(C_i), i) - radius(C_i) \tag{6}$$

Combining the precondition and the inequality. Equation (6), we can deduce that

$$dist(u, i) > maxdknn(C_i) \geq maxdknn(u) \tag{7}$$

Based on the calculation above, we can confirm that when $dist(i, center(C_j)) - radius(C_j) > maxdknn(C_j)$, the users in C_j might not be influenced by the updating of i, therefore, C_j can be pruned. □

The difference between the applied pruning condition and the full-dimensional pruning condition on HDR Tree is gained by using the left side of one condition to reduce that of the other condition and then taking the absolute value that $|dist(i, center(C_j)) - dist_{PCA}(i, center(C_j))| + |radius_{d=d(C_j)}(C_j) - radius(C_j)|$. This difference can be used to quantify the loss of spatial features caused by PCA; it is inversely proportional to the number of dimensions reserved by PCA. The greater the loss of spatial information, the greater the possibility of Type I error and the lower the pruning efficiency.

In conclusion, when applied to a non-globally correlated dataset, the searching efficiency of a single HDR Tree can be inadequate. The performance of a single HDR Tree is hindered by its inability to effectively prune unaffected users at low levels, which consequently necessitates increased distance calculations to traverse deeper levels. To illustrate, we present an example of a globally correlated dataset and a non-globally correlated but locally correlated dataset.

Example 3 In Figure 2, the left sub-figure shows a globally correlated dataset, where the variance on the first dimension of the vectors produced by PCA projection accounts for a dominant proportion. However, more dimensions from the global PCA matrix are required to capture the global spatial features in the right sub-figure, whereas C_1 and C_2 are locally correlated clusters whose spatial features can be obtained using only a few dimensions from local PCA. A single HDR Tree will perform weaker on the dataset in the right sub-figure because many unaffected users cannot be pruned early.

Expensive operations on a large tree Furthermore, apart from the inadequate pruning capabilities resulting from the presence of non-globally related data, there exists another concern associated with the utilisation of a single HDR Tree. As the volume of data expands, the size of the tree also grows, causing a single HDR Tree to become very long on a large dataset, which leads to expensive searching and updating.

6.1 HDR forest

To overcome the limitations of using a single HDR Tree, we introduce the concept of the HDR Forest (HDRF). The HDRF comprises multiple local HDR Trees, built on the *local* vectors by the *local* PCA matrices. Strong local correlation allows for more unaffected users to be pruned at lower levels of the local HDR Trees, saving the distance calculation required to search deeper in the trees. Therefore, an HDR Forest can implement faster RkNN searching

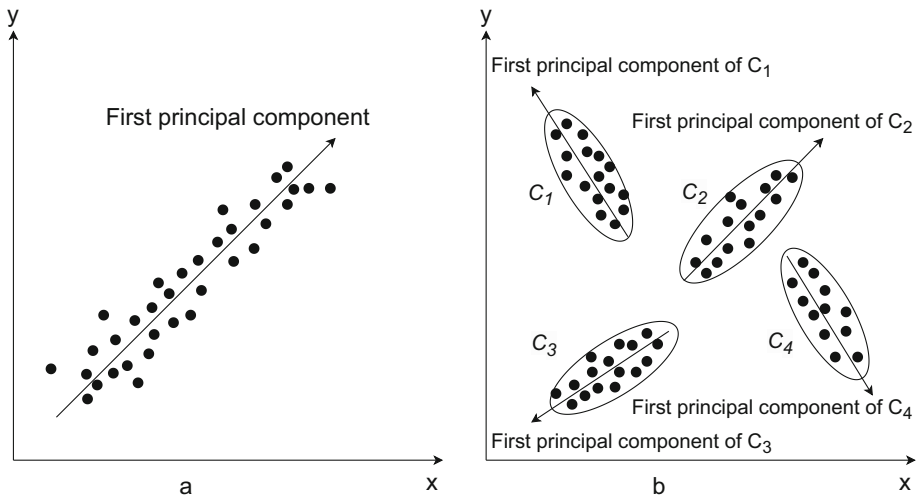

Figure 2 Correlation variance illustration of PCA

than a single HDR Tree when the dataset is not globally correlated but locally correlated. Furthermore, because only a few local HDR Trees of small scale instead of a global HDR Tree of large scale are searched and updated when an item is updated, the time cost of a RkNN search can be further decreased, and the time cost for updating the data structure can be decreased. This subsection's content explains the structure and construction process of HDR Forest, as well as the RkNN search on HDR Forest.

Structure of HDRF A HDRF($n \times f$) consists of $n \times f$ local HDR Trees $\{T_{1,1}, \ldots, T_{1,f}, \ldots, T_{i,1}, \ldots, T_{i,f}, \ldots, T_{n,1}, \ldots, T_{n,f}\}$ where n and f are decided by the programmers. $T_{i,j}$ means the local HDR Tree built on the users in $S_{i,j}$ by the local PCA of the user vectors in $S_{i,j}$. Note that $S_{i,j}$ is the j-th section of the i-th cluster of the whole data.

A local HDR Tree $T_{i,j}$ contains the following information: the tree structure of $T_{i,j}$, $maxdknn(S_{i,j})$, $center(C_i)$, $do_{i,j}$ and $di_{i,j}$. The maximal distance from the user vectors in $S_{i,j}$ to the center of the i-th cluster is labelled as $do_{i,j}$. The minimal distance from the user vectors in $S_{i,j}$ to the center of the i-th cluster is labelled as $di_{i,j}$.

Example 4 An example of the structure of an HDRF(4×2) is shown in the Figure 3

Construction of HDRF An HDR Forest is constructed by following the steps outlined below. The algorithm for the construction process of an HDRF is shown in Algorithm 5.

Step 1. Cluster the entire user set into n clusters, labelled $C_1, \ldots, C_n$, using a specific method. We use the k-means clustering method in this paper to make things easier.

Step 2. Based on the distance between the vectors and the center of cluster C_i, divide every C_i cluster of the n acquired clusters into f sections $S_{i,1}, \ldots, S_{i,f}$. Any user vector u_k in $S_{i,j}$ must meet the following requirements: $(j-1) \times radius(C_i)/f < dist(u_k, center(C_i)) \leq j \times radius(C_i)/f$.

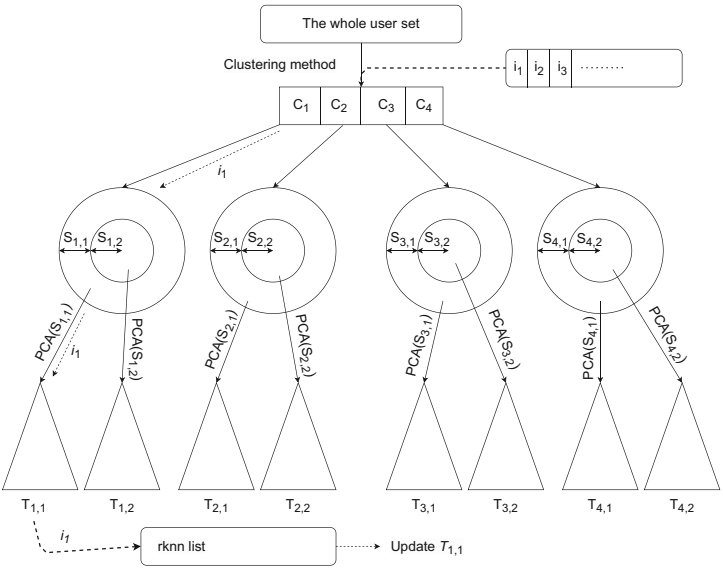

Figure 3 The structure and RkNN process of the HDR Forest

Step 3. For each section $S_{i,j}$, calculate the PCA matrix of the user vectors in $S_{i,j}$ and construct the corresponding local HDR Tree $T_{i,j}$ based on the computed local PCA matrix, record the distance between the outermost user in $S_{i,j}$ and the center of C_i as $do_{i,j}$ and the distance between the innermost user in $S_{i,j}$ and the center of C_i as $di_{i,j}$. The information of $do_{i,j}, di_{i,j}, center(C_i), maxdknn(S_{i,j})$ and the local PCA matrix of $S_{i,j}$ are recorded in $T_{i,j}$.

Algorithm 5 Construction of HDR forest.

Require: The whole user set U, the number of local clusters n, the number of sections that each local cluster is divided into f, the fanout for each HDR Tree F, the threshold of each HDR Tree θ
Ensure: A HDR Forest
Initialise an empty list L
C=Clusters(U, n)
for each C_j in C **do**
  $r \leftarrow radius(C_j)/f$
  for int i in [1, f] **do**
    Initialise an empty list l
    Initialise an empty list l_1
    Initialise an empty list l_2
    for each user u in C_j **do**
      if $(f-1) \times r < dist(u, center(C_j)) \leq f \times r$ **then**
        push u in l
        push $dist(u, center(C_j))$ in l_1
        push dknn(u) in l_2
      end if
    end for
    Initialise a Local HDR Tree T
    $T.maxdknn \leftarrow \max(l_2)$
    $T.d_o \leftarrow \max(l_1)$
    $T.d_i \leftarrow \min(l_1)$
    $T.matrixpca \leftarrow$ CalculatePCA(l)
    $T.tree \leftarrow$ ConstructHDR(l, F, θ)
    $T.center \leftarrow C_j.center$
    push T in L
  end for
end for
return L

RkNN process by HDRF When an item I is updated, all the local HDR Trees in the HDRF structure will be traversed. For a visited local HDR Tree $T_{i,j}$, a pruning condition that $(dist(I, center(C_i)) > do_{i,j} + maxdknn(S_{i,j})) \vee (dist(I, center(C_i)) < di_{i,j} - maxdknn(S_{i,j}))$ will be checked; if the condition is satisfied, $T_{i,j}$ can be pruned and does not need to be searched, otherwise, search should be implemented on it and $T_{i,j}$ should be updated. Note that an example and an explanation of the pruning condition are presented below this paragraph. A detailed description of the pruning condition is in Theorem 4. The process of RkNN search by HDRF is depicted in Figure 3. The detailed process of RkNN search by HDRF is illustrated in Algorithm 6.

Example 5 To provide a concrete illustration, consider Figure 4. In this scenario, an RkNN search on the $S_{i,j}(T_{i,j})$ is only required when the updated item I is in the available region for the$S_{i,j}(T_{i,j})$.

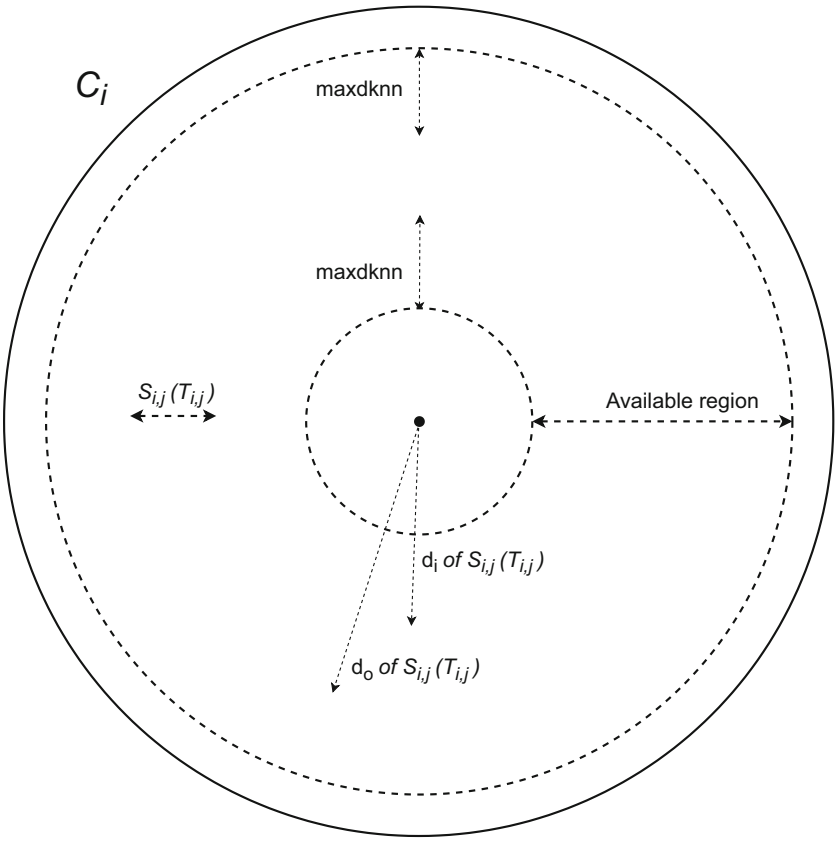

Figure 4 The RkNN search only needs to be performed when an item is in the available region of the particular section (local HDR Tree)

Theorem 4 (Pruning on HDR Forest for RkNN search) *If* $(dist(I, center(C_i)) > do_{i,j} + maxdknn(S_{i,j})) \vee (dist(I, center(C_i)) < di_{i,j} - maxdknn(S_{i,j}))$ *when a local HDR Tree* $T_{i,j}$ *is visited during the RkNN search of an updated item* i *on an HDRF, the users in the local HDR Tree* $T_{i,j}$ *might not be influenced by the updating of* i*; therefore,* $T_{i,j}$ *can be pruned and does not need to be searched.*
Otherwise, the users in the local HDR Tree $T_{i,j}$ *should be searched.*

Proof If $do_{i,j} < dist(I, center(C_i)) - maxdknn(S_{i,j})$, then for an arbitrary user u in the section $S_{i,j}$, $dist(I, u) \geq dist(I, center(C_i)) - dist(u, center(C_i)) \geq dist(I, center(C_i)) - do_{i,j} > dist(I, center(C_i)) - (dist(I, center(C_i)) - maxdknn(S_{i,j})) = maxdknn(S_{i,j})$. As a result, the user vectors in the section $S_{i,j}$ (in the local HDR Tree $T_{i,j}$) can be pruned.
If $di_{i,j} > dist(I, center(C_i)) + maxdknn(S_{i,j})$, then for an arbitrary user u in the section $S_{i,j}$, $dist(I, u) \geq dist(u, center(C_i)) - dist(I, center(C_i)) \geq di_{i,j} - dist(I, center(C_i)) > dist(I, center(C_i)) + maxdknn(S_{i,j}) - dist(I, center(C_i)) = maxdknn(S_{i,j})$. As a result, the user vectors in the section $S_{i,j}$ (in the local HDR Tree $T_{i,j}$) can be pruned. □

Algorithm 6 RkNN search on HDR forest.

Require: an item I that is deleted from or inserted into the sliding window, an HDRF structure with a list L containing multiple local HDR Trees
Ensure: The users affected by I
1: Initialise an empty list R
2: **for** T in L **do**
3: **if** $T.d_i - T.maxdknn \leq dist(I, T.center) \leq T.d_o + T.maxdknn$ **then**
4: RkNN_Search(R, $T.tree$)
5: Update(T)
6: **end if**
7: **end for**
8: return R

6.2 Precomputation and index of PCA

This subsection optimises RkNN search on HDR Tree by adding an additional dynamic list indexing the PCA states of the updated item. Because of the similarity between the structure of HDR Tree and the structure of local HDR Tree among HDR Forest, such a supportive list is also available for HDR Forest. For convenience, in the following content explaining the precomputation and index of the PCA states of the updated item, we assume that the dynamic index list is used with an HDR Tree. The application of the dynamic index list on HDR Forest is described at the end of this subsection.

Repeated PCA projection HDR Tree has a flaw that it does not index the low-dimensional states of the item vectors produced by PCA projection. (In the following content, "PCA states" works as a substitute for "low-dimensional states" because the low-dimensional states of an item are produced by PCA projection from that item.) This lack of index will result in repeated PCA projection when multiple clusters on the same level of the HDR Tree are visited. Below is an example of repeated PCA projection.

Example 6 For example, as is shown in Figure 5, when the child node of the cluster C_f is visited during the RkNN search of an updated item i, $PCA_{d(2)}(i)$ must be calculated twice for the calculation of $dist_{PCA_{d(2)}}(i, center(C_{f1}))$ and $dist_{PCA_{d(2)}}(i, center(C_{ff}))$ if there is no index for the PCA states of i.

Precomputation and index of PCA In order to resolve the aforementioned issue, we include a dynamic list in the HDR Tree that indexes the PCA states of the updated item. When an item i is updated, the list is cleared first, and then $PCA_{d(l)}(i)$ is precomputed and stored in the list for each level l on the HDR Tree. When processing a cluster C_j on HDR Tree level l, $PCA_{d(l)}(i)$ can be extracted directly from the dynamic PCA index list rather than implementing PCA projection. This PCA index list helps to address the issue of repeating

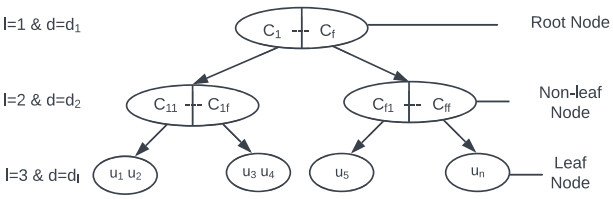

Figure 5 The structure of HDR Tree, where $fanout = f$

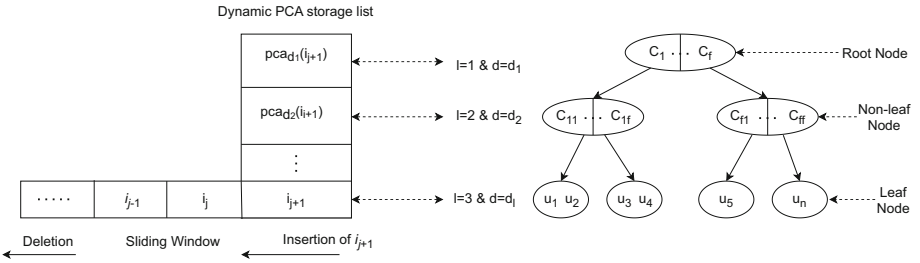

Figure 6 The HDR Tree structure with a dynamic list indexing the PCA states of the updated item

PCA projection when processing several clusters at the same level. The structure of the HDR Tree combined with the dynamic list indexing the PCA states of the updated item is as shown in Figure 6.

Use of precomputation and index of PCA on HDRF If RkNN search for an updated item i is implemented on a HDRF supported by a dynamic PCA index list, the dynamic list is cleared at the beginning stage of visiting an arbitrary local HDR Tree $T_{i,j}$ among the HDRF. Then the PCA states of i will be calculated by the partial local PCA matrices of the user vectors in $S_{i,j}$. After that, the calculated PCA states will be stored in the dynamic list. As a result, $PCA_{d(l)}(i)$ can be extracted directly from the list rather than calculation of PCA projection when calculation of the low-dimensional distance between a cluster on level l of $T_{i,j}$ and i.

6.3 Pruning-based kNN recomputation

This subsection focuses on enhancing the HDR Tree by utilising a novel algorithm designed to efficiently recalculate the kNN lists of affected users when items are deleted from the sliding window. This method, referred to as "pruning-based kNN recomputation" proves to be highly effective. Furthermore, its applicability extends to HDRF as a means to accelerate updating affected kNN lists. For clarity, we assume in the following description of pruning-based kNN recomputation that the algorithm is applied to a HDR Tree. And the application of pruning-based kNN recomputation on HDRF is discussed at the end of this subsection. Here, for a better understanding of the following content, new symbols used in the following content and their meanings are listed in Table 2 instead of Table 1.

Disadvantages of basic kNN recomputation When an item is deleted from the sliding window, an affected user u_a must search for its new k-th nearest item from W_u. The conven-

Table 2 New symbol and definitions

Symbols	Definitions
W_o	Sliding Window before an item is deleted
W_u	Sliding Window after an item is deleted
R_o	kNN set before the item is deleted
R_u	kNN set after the item is deleted
$knnf$	A set whose elements are the friend user vectors

tional search approach involves computing the distance between u_a and the items in $W_u - R_o$ to confirm the nearest item to u_a in $W_u - R_o$. Note that the distance between u_a and the items in $W_u \cap R_o$ needs not to be calculated because such items have been in the kNN set of u_a. And it is obvious that the nearest item in $W_u - R_o$ is the new k-th nearest item. Such basic method for updating kNN lists is highly time costly because distance calculation in this process is on full dimensions. Based on the observation above, we propose the pruning-based kNN recomputation for updating the kNN lists of the affected users when deletion of items. Before describing the algorithm, we introduce the concept of "friend user" which is defined in Definition 5.

Definition 5 (Friend User) A user u_x is a friend user of u_y in an HDR Tree T (local), if and only if there exists a leaf node LF that $LF \in T \wedge u_x, u_y \in LF$.

Preparation of pruning-based kNN recomputation In order to accelerate the computation, we precompute the PCA states of the users and items whose number of dimensions can be chosen by the appliers but should be limited under a small scale to confirm fast distance calculation. And then, for each user and item, its computed PCA state will be stored and attached to it. Supported by the storage of PCA states of the user vectors and the item vectors, the time cost for projecting the full-dimensional vectors to the PCA space can be saved every time $dist_{PCA}(u, i)$ needs to be computed.

Algorithm 7 Update kNN list.

Require: An affected user u and a deleted item I
Ensure: The updated kNN list of u
1: Initialise two empty lists L and T
2: **for** u_i in LN that contains u **do**
3: **for** i_j in $u_i \rightarrow R$ **do**
4: **if** not i_j in $u \rightarrow R$ **then**
5: push $dist(i_j, u)$ in L
6: **end if**
7: **end for**
8: **end for**
9: $B \leftarrow min(L)$
10: **for** i_k in sliding window **do**
11: **if** not i_k in $u \rightarrow R$ **then**
12: **if** $distance_{PCA}(i_k, u) \leq bound$ **then**
13: push$(i_k, distance(i_k, u))$ in T
14: **end if**
15: **end if**
16: **end for**
17: Get the item i with its distance to u least in T
18: Push i into $u \rightarrow R$
19: Delete I from $u \rightarrow R$

Explanation of the algorithm The algorithm is described in Algorithm 7. In the first stage, two empty lists L and T are initialised in line 1, where L is for storing the distance between the affected user u_a and the items in $knnf(u_a) - R_o(u_a)$ and T is for storing the tuple composed of i_k and $dist(i_k, u_a)$ for each item in a subset of $W_u - R_o(u_a)$ which is produced by pruning a large proportion of items in $W_u - R_o(u_a)$. And the operation in lines 2-9 gains the minimal distance between u_a and the items in $knnf(u) - R_o(u_a)$ and stores that shortest distance as

a pruning bound B.(Note that if a friend user u_b of u_a is also affected, then either $R_u(u_b)$ or $R_o(u_b)$ is available in this step.) And through the loop with if statements in lines 10-16, the nearest item to u_a among the items that are in $W_u - R_o(u_a)$ and have their low-dimensional distance to u_a no larger than B is gained. The updating process of $R(u_a)$ is completed by the operation in lines 17-19 which pushes the gained item in $R(u_a)$ and deletes I which is the item deleted from the sliding window from $R(u_a)$.

Example 7 For instance, consider the situation shown in Figure 7 where 5NN is applied. When i_4 is deleted from the sliding window, the original 5NN list of an affected user u_3 is i_1, i_2, i_3, i_4, i_5 and the 5NN list of its only one friend user u_4 is i_3, i_5, i_6, i_7, i_8, then, the pruning bound B for u_3 should be the minimum among the distance between u_3 and i_6, i_7, i_8. And for each item i_k in $W_u - R_o(u_3)$, only when $dist_{PCA}(i_k, u_3) \leq B$ will i_k be considered as the potential new 5-th nearest neighbor item of u_3.

Complexity analysis The advantage of Pruning-based kNN Recomputation is that it only requires distance calculation of low-dimensional for most of the items in $W_u - R_o$. The friend users of the affected user u_a are very close to u_a because they are assigned to the same leaf node with u_a after hierarchical clustering, as a result, the closest item to u_a among the items in $knnf(u_a) - R_o(u_a)$ is also very close to u_a among the items in $W_o - R_o(u_a)$ and can provide a very tight pruning bound B. For an item i_k in $W_u - R_o(u_a)$, it is highly possible that $dist(i_k, u_a) > B$ because of the tightness of the pruning bound B. As a result, it is highly possible that $dist_{PCA}(i_k, u_a) > B$ because $dist(i_k, u_a) - dist_{PCA}(i_k, u_a)$ can be very small due to the feature of PCA. Therefore, it is highly possible that i_k can be pruned from the candidate list before the calculation of distance on full dimension begins. In conclusion, only a few items in $W_u - R_o(u_a)$ require distance calculation on full dimensions.

Correctness The following discussion proves that Pruning-based kNN Recomputation offers the affected user u_a the nearest item in $W_u - R_o(u_a)$ which is also the new k-th nearest item for u_a. Let's categorise the items in $W_u - R_o(u_a)$ into two groups. The first group is labelled as G_1. We have $G_1 = \{i_1 | i_1 \in W_u - R_o(u_a), dist_{PCA}(i_1, u_a) > B\}$. The second group is labelled as G_2. We have $G_2 = \{i_2 | i_2 \in W_u - R_o(u_a), dist_{PCA}(i_2, u_a) \leq B\}$. It becomes evident that for an arbitrary item i, if $i \in G_1$, then $\exists i_B, i_B \in W_u - R_o(u_a)$ and $dist(i_B, u_a) < dist(i, u_a)$. Because the item in $knnf(u_a) - R_o(u_a)$ providing the bound

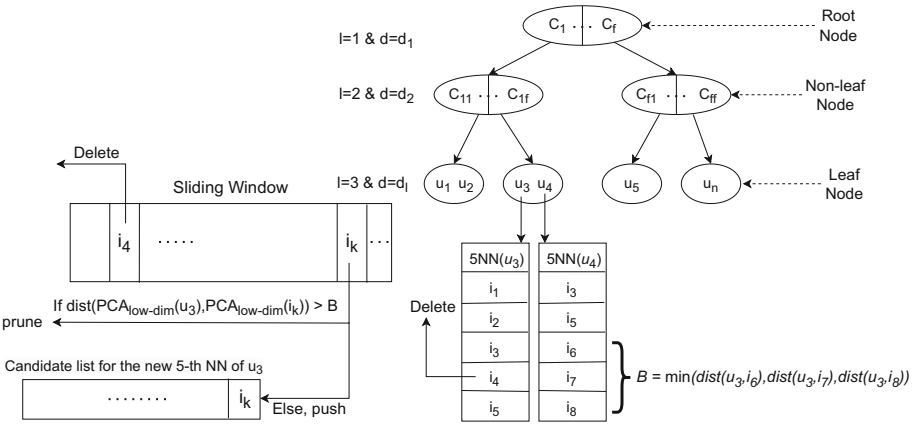

Figure 7 The pruning policy when optimised deletion

B can serve as i_B because of the inequality $B < dist_{PCA}(i, u_a) \leq dist(i, u_a)$. As a result, any given item in G_1 cannot be the closest item to u_a in $W_u - R_o(u_a)$. Therefore, for an item i_m, if $dist(i_m, u_a) = min\{dist|\exists i \in G_2, dist(i, u_a) = dist\}$, then $dist(i_m, u_a) = min\{dist|\exists i \in W_u - R_o(u_a), dist(i, u_a) = dist\}$. The result provided by Pruning-based kNN Recomputation satisfies the condition above and is in $W_u - R_o(u_a)$, consequently, Pruning-based kNN Recomputation provides the nearest item to u_a in $W_u - R_o(u_a)$ and the new k-th nearest item for u_a.

Pruning-based kNN recomputation in HDR forest When Pruning-based kNN recomputation is used to update the kNN list of an affected user u on HDRF, the friend users are the users that are in the same leaf node with u in a local HDR Tree.

7 Performance evaluation

In this section, we present our experimental results. All experiments are conducted on a computer with an Intel Core i5-4210U 2.4GHz processor, 12GB RAM, and Windows 10 OS. All methods are implemented in C++.

7.1 HDR$^+$ tree

In this subsection, we compare our HDR$^+$ Tree (Section 4) with the two baseline algorithms:

1. *NaiveRkNN*: The naive approach of searching affected users without an index. It computes the distance between the item and all users within U and decides whether or not to update the user recommendation list.
2. *HDR Tree*: The HDR Tree [26] method that searches for the affected users caused by any update operation as discussed in Section 4.

We performed most of the experiments on the 128 Dimensional NUS-WIDE Image DataSet [55], which consist of 269, 648 records from the Flickr dataset. A 128-dimensional dataset is used by default. We build the sliding window, which acts as an item stream. The default sliding window W size is kept at 200, 000, and 50, 000 random users are selected. We set the default value of k to 10 and the fanout f to 5. The default number of updates is set to 100 following [26].

Exp-1: Varying the number of updated items We compare the effect of updated items by varying it from 100 to 600. Figure 8 shows that the cost for searching the affected items is linearly increasing for all the approaches, but the batch update and lazy update outperform the baseline approaches. The batch and lazy updates are about 1.5 and 3 times faster than the HDR Tree.

Exp-2: Varying W and k In Figure 9, it can be seen that when increasing the k size, the elapsed time increases in all the approaches. But lazy updates and batch updates give better results than baseline approaches. When we increase the k size, the $maxdknn$ also grows, resulting in more affected users. As shown in Figure 10., with an increase in the sliding window size W, the time cost increases for all the approaches as the computation cost increases.

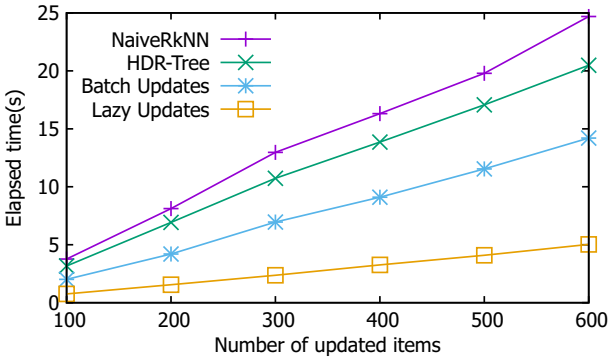

Figure 8 Vary number of updated items

Exp-3: Varying the number of features We conduct experiments on other datasets with different dimensions in the NUS-WIDE Image DataSet collection. The dimension varies from 128 to 500. Figure 11 illustrates that with the increase of dimensionality, the execution time increases because the time required for distance computation increases. As shown in the figure, our approaches outperform existing ones.

Exp-4: The effect of deletion optimisation We compare the effectiveness of our deletion optimisation with the naive HDR Tree in this experiment. For the HDR Tree method, we use the HDR Tree to find the RkNN list of a being deleted item. In our method (denoted as the RkNN Table in Figure 12), we directly extract the RkNN list of a deleted item from the RkNN table. Note that we normalise the time cost as the ratio between the real-time cost for updating items and the time cost for updating 100 items using our method. As observed, our method achieves about 15% of improvement when updating 600 items.

7.2 HDR forest

In this subsection, we compare four versions of the HDR Forest (Section 6):

1. *HDR Tree*: A single HDR Tree without any optimisation can be regarded as an HDR Forest of 1×1 size.

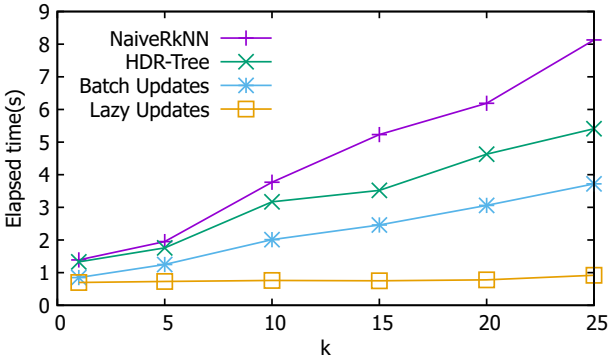

Figure 9 Vary the number of k

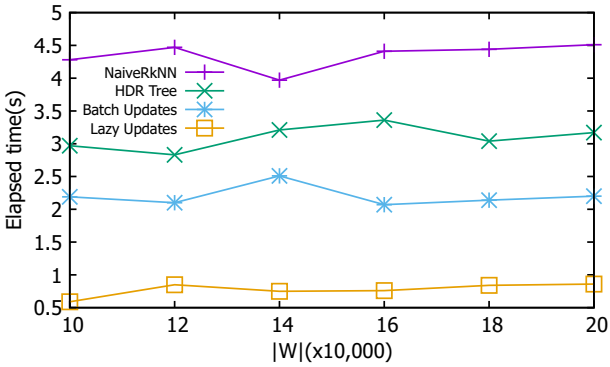

Figure 10 Vary the number of W

2. *HDR Forest*: A forest consists of multiple local HDR Trees built by the PCA matrices of the local data. The HDR Forest uses local correlation to accelerate the RkNN search process, whether it's for deleted or newly inserted items in the sliding window.
3. *HDR Forest with Precomputation and Indexing of PCA*: Precomputation and Indexing of PCA can further optimise the process of RkNN search on HDR Forest by precomputing and indexing the PCA states of the updated item by local PCA matrices of the user vectors that form the local HDR Trees. (Detailed information on the PCA precomputation and indexing is in Section 6.2). In this subsection, we denote HDR Forest supported by Precomputation and Indexing of PCA as HDR$^+$ Forest.
4. *HDR Forest with Precomputation and Indexing of PCA and with Pruning-based kNN Recomputation*: When both Precomputation and Indexing of PCA and Pruning-based kNN Recomputation are added to the HDR Forest, both the process of updating the affected kNN lists and the process of RkNN search on HDR Forest can be accelerated. (Detailed information on Pruning-based kNN Recomputation is in Section 6.3). In this subsection, we denote HDR Forest supported by Precomputation and Indexing of PCA and by Pruning-based kNN Recomputation as HDR* Forest.

We reserve most of the experiment settings of the experiments for HDR$^+$-Tree. The default sliding window W size is kept at 50, 000, and the default number of updates is set to 5, 000.

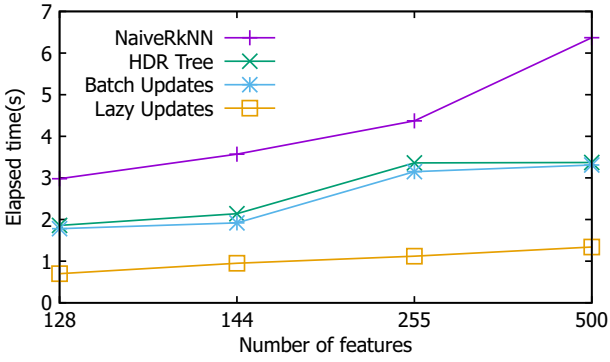

Figure 11 Vary the number of features

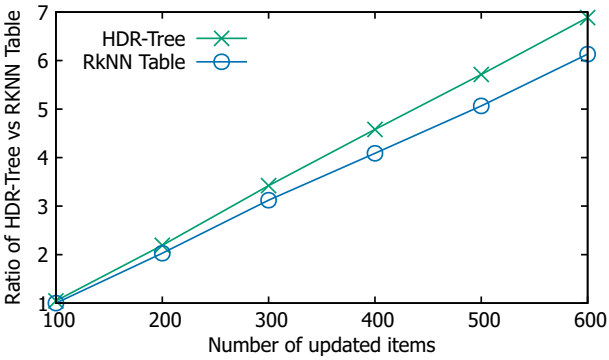

Figure 12 HDR Tree vs RkNN table in deletion

Note that HDR Forest with Precomputation and Indexing of PCA and with Pruning-based kNN Recomputation will not be implemented in the experiments for insertion of items because the algorithm Pruning-based kNN Recomputation is designed to recompute the affected kNN lists when item deletion instead of insertion.

Exp-5: Varying the insertion number We detect the influence of the number of inserted items on the advantage of HDR Forest and HDR$^+$ Forest compared with basic HDR Tree by varying the number of insertions from 1000 to 10000. Figure 13 illustrates that the time consumption for all the methods and the insertion number form a positive linear relation. However, the HDR Forest performs better than the basic HDR Tree by 12%-shorter time cost. Besides, HDR Forest can be strengthened by about 11.5% if combined with Precomputation and Indexing of PCA.

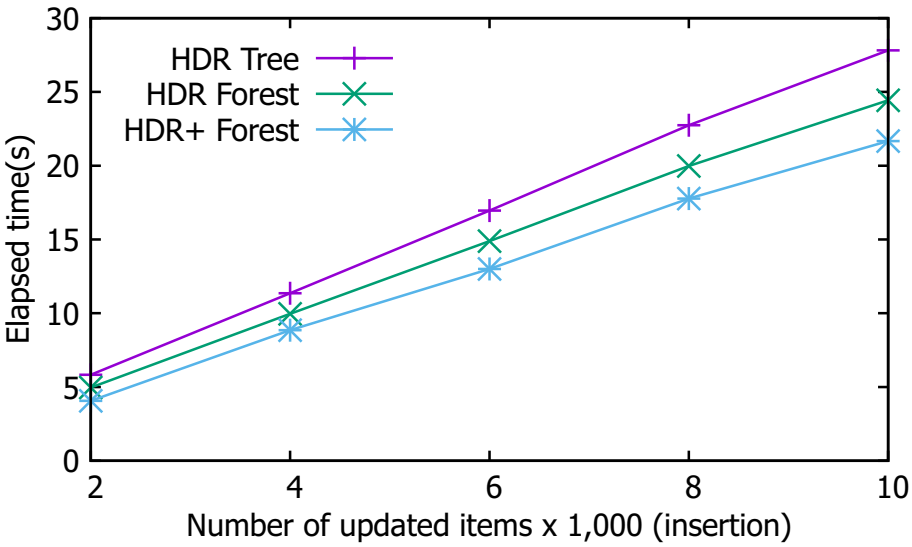

Figure 13 Vary number of updated items for insertion

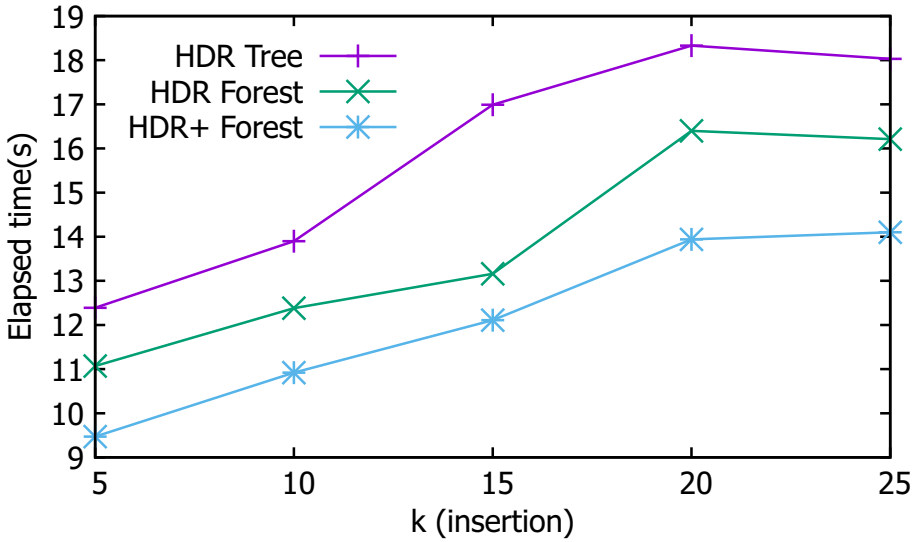

Figure 14 Vary the number of k for insertion

Exp-6: Varying k when insertion We study the impact of varying k from 5 to 25 on the performance difference between HDR Forest, HDR$^+$ Forest, and basic HDR Tree. Figure 14 shows that as k increases, the time consumption of all methods increases. When k increases, the $dknn$ value of the users will increase so that more users might be affected by the inserted items. As a result, it needs more time to do RkNN search to search for the affected users. HDR Forest still keeps its advantage of about 10% towards basic HDR Tree, and Precomputation and Indexing of PCA can enhance HDR Forest by about 12%.

Exp-7: Varying number of dimensions when insertion According to Figure 15, an increase in dimensions from 32 to 160 increases the time cost of all the methods except for the situation where the number of dimensions is increased from 96 to 128. This is because the time complexity of distance calculation is of positive relation with the number of dimensions. And when the data is 128D, the distribution of the dataset might be special that fewer users

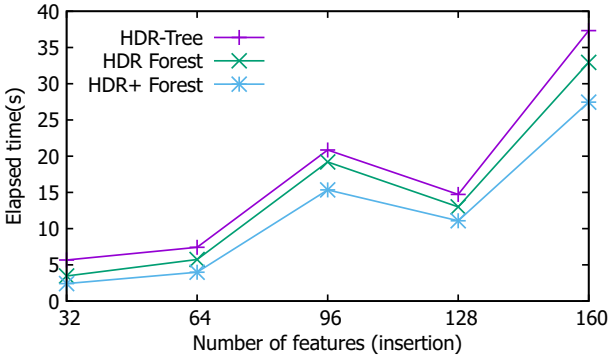

Figure 15 Vary the number of features for insertion

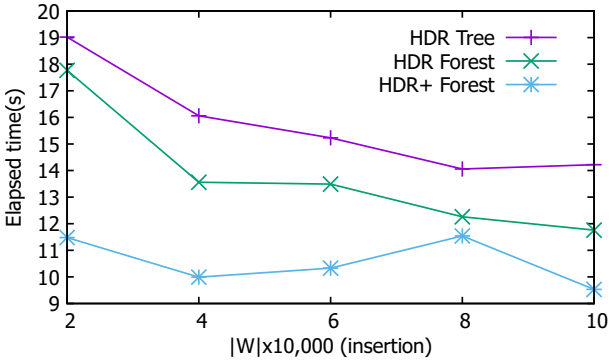

Figure 16 Vary the number of W for insertion

are influenced by the inserted items than in the 96D dataset. As a result, RkNN search time is an anomaly. HDR Forest still keeps its average advantage of about 13% towards basic HDR Tree, and Precomputation and Indexing of PCA can contribute about 15% acceleration when combined with HDR Forest.

Exp-8: Varying window size when insertion In Figure 16, the trend of time cost for all the methods appears to be generally descending when the window size is from 20,000 to 100,000. This is because when the window size increases, the $dknn$ value of the users decreases. As a result, less users will be influenced by the inserted items. Therefore, less influenced users need to be found, and the RKNN Search process costs less time. HDR Forest is about 12% faster than basic HDR Tree, and HDR$^+$ Forest outperforms HDR Forest by about 15%.

Exp-9: Vary user set size when insertion In Figure 17, when the user set is enlarged from 20,000 to 100,000, the time cost for all the methods increases because more users have to be searched, and more users are affected. And larger user set might lead to a weaker global correlation, which might be the reason that the advantage of HDR Forest toward HDR Tree becomes more obvious when the user set is larger. And more searched users lead to more visiting of the clusters on the same level of local HDR Trees, so the function of Precomputation

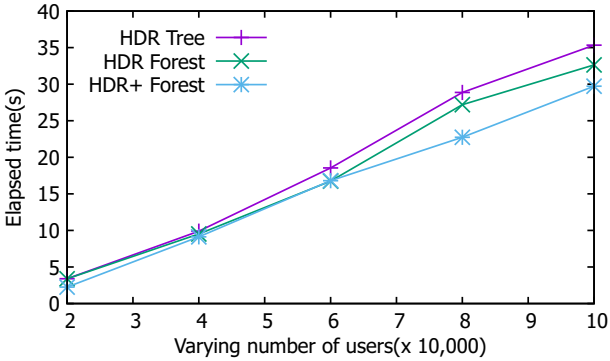

Figure 17 Vary number of user set for insertion

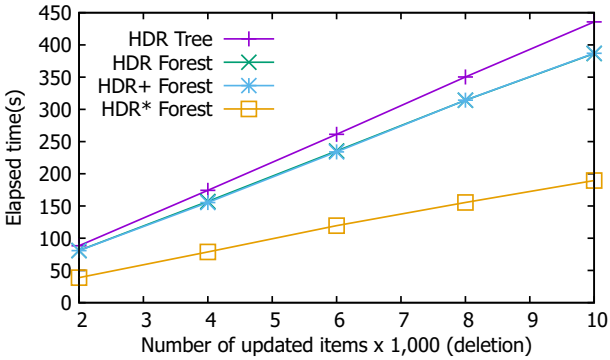

Figure 18 Vary number of updated items for deletion

and Indexing of PCA is more obvious. HDR Forest is about 12% faster than basic HDR Tree, and HDR$^+$ forest is faster than HDR Forest by about 13%.

Exp-10: Varying number of updated items when deletion Figure 18 shows that as the number of deleted items increases from 1000 to 10,000, the time cost of dynamic kNN join by all the methods increases linearly. HDR* Forest costs less than 50% time of other methods. That is because when item deletion, updating of the affected kNN lists costs much more time than the RkNN search. The time cost for updating the affected kNN lists is dominant in the whole time for dynamic kNN join when deletion. And HDR* Forest makes full use of the pruning technique and dimensionality reduction to accelerate kNN updating effectively when item deletion, while the other methods only use the basic method to update the affected kNN lists. So HDR* Forest shows a huge advantage over other methods. The reason why the difference between HDR Forest and HDR Tree becomes larger is that when some items are deleted from the sliding window, $dknn$ value of the users will become larger. As a result, more users are influenced by deleted items to be found by RkNN search. Therefore, the advantage on RkNN search of HDR Forest towards HDR Tree is enlarged. And the difference between HDR Forest and HDR$^+$ Forest seems trivial because both HDR Forest and HDR$^+$ Forest use the basic method to update kNN lists, and the time difference between such two methods is

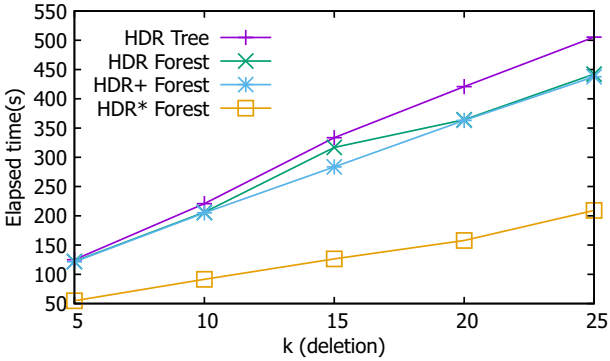

Figure 19 Vary the number of k for deletion

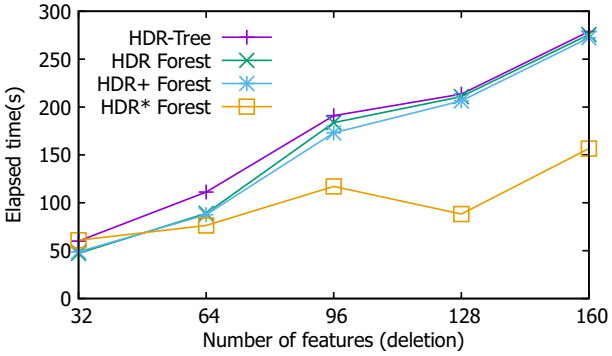

Figure 20 Vary the number of features for deletion

their time difference on RkNN search, which might be almost covered by the relatively long kNN updating time.

Exp-11: Varying *k* when deletion As is shown in Figure 19, when the number of k increases from 5 to 25, the time cost of dynamic kNN join by all the methods increases. HDR* Forest costs less than 50% time of other methods because of its efficient updating of affected kNN lists when deletion. The reason why the difference between HDR Forest and HDR Tree becomes larger is that when k increases, the $dknn$ value of the users becomes larger, which means more affected users by deleted items to be found. Therefore, the advantage on RkNN search of HDR Forest towards HDR Tree is enlarged. And the difference between HDR Forest and HDR$^+$ Forest seems trivial because the time difference on the RkNN search is almost covered by the relatively long kNN updating time.

Exp-12: Varying number of dimensions when deletion As is shown in Figure 20, the time for dynamic kNN join increases when the number of dimensions increases from 32 to 160, regardless of the method used, during deletion. And the efficiency of HDR* Forest is significantly higher compared to other methods by about 1.6% to 60%. This advantage is magnified as the number of dimensions increases. That is because when the number of

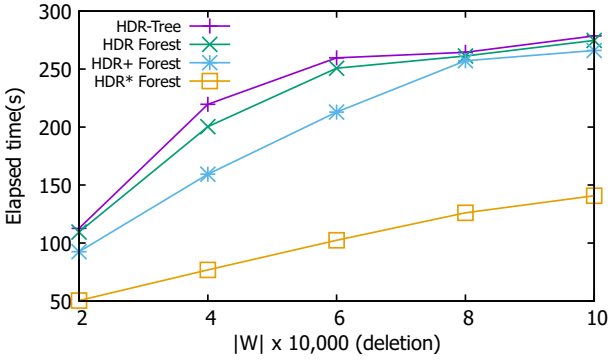

Figure 21 Vary the number of W for deletion

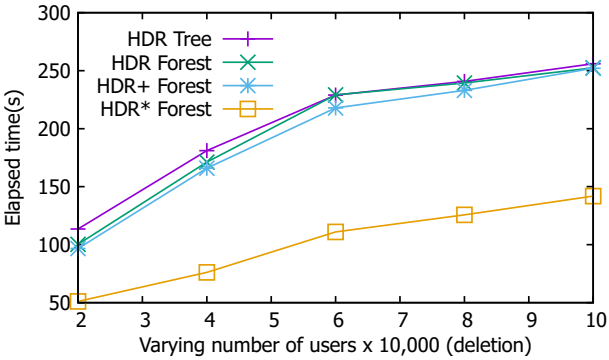

Figure 22 Vary number of user set for deletion

full dimensions go upward, the difference between the time for distance calculation on full dimensions and that on low dimensions becomes larger.

Exp-13: Varying window size when deletion According to Figure 21, an increase in window size from 20,000 to 100,000 prolongs the time of dealing with item deletion, no matter which method is used. This is because when the window size increases, more distance calculation between the affected users and the items in the sliding window is needed to update kNN lists. However, HDR* Forest still reserves its advantage by about 50% over other methods. The time difference between HDR Tree and HDR Forest and HDR$^+$ Forest becomes smaller when the window size increases. This is because a larger window size results in a smaller $dknn$ value for users, which means that there are less affected users to be found by RkNN search. So the advantage of RkNN search brought by Precomputation and Indexing of PCA and local correlation seems less obvious.

Exp-14: Varying size of user set when deletion According to Figure 22, when the size of the user set increases from 20,000 to 100,000, time consumption for all the methods increases. This is because when the user set is extended, more users might be affected by the deleted items, which leads to more distance calculations to update the kNN lists of such users. When the user set size increases, the speed of using HDR* Forest to deal with deletion is about 50% faster than other methods.

8 Conclusion

In this paper, we study the problem of continuous kNN join over dynamic high-dimensional data. We first propose the HDR$^+$ Tree based on the HDR Tree, which has the functions of efficient insertion, deletion, and batch update. Moreover, we propose the HDR Forest to capture the local features of non-globally correlated datasets. Two optimisations, namely precomputation of PCA and pruning-based kNN recomputation, are further introduced to address the limitations of redundant PCA projection and inefficient recomputation of the affected KNN lists during deletion. Furthermore, we prove the theorem that distance in PCA space is always no larger than the distance in its full dimensionality. Our experiments on

real-world datasets demonstrate that our approaches achieved high performance for dynamic high-dimensional data and significantly outperform the existing approaches.

Author Contributions Conceptualization, N.U., G.Z., Z.Y., and W.Z.; methodology, N.U., G.Z., Z.Y.; validation, N.U., G.Z., Z.Y., B.L., W.L., and W.Z.; formal Analysis, N.U., G.Z., Z.Y., and W.L.; resources, Z.Y., B.L., W.L., and W.Z.; coding, N.U. and G.Z.; writing—original draft preparation, N.U. and G.Z.; writing—review & editing, N.U., G.Z., and Z.Y.; proofreading N.U., G.Z., Z.Y., B.L., W.L., and W.Z.; supervision, Z.Y., B.L., W.L., and W.Z.; project administration, B.L. and W.Z.; funding acquisition, B.L. and W.Z.

Funding Wei Li is supported by the Natural Science Foundation of Heilongjiang Province (LH2023F020), and the National Natural Science Foundation of China (62272126).

Availability of data and materials We have used the open-source NUS-WIDE Image Dataset [55].

Declarations

Conflicts of interest The authors declare no conflict of interest.

Ethics approval Not applicable.

References

1. Dasarathy, B.V.: Nearest neighbor (nn) norms: Nn pattern classification techniques. IEEE Computer Society Tutorial (1991)
2. Zhang, S., Li, X., Zong, M., Zhu, X., Wang, R.: Efficient knn classification with different numbers of nearest neighbors. IEEE Trans. Neural Netw. Learn. Sys. **29**(5), 1774–1785 (2017)
3. Zhou, C., Tham, C.-K.: Graphel: A graph-based ensemble learning method for distributed diagnostics and prognostics in the industrial internet of things. In: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 903–909 IEEE (2018)
4. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. J. R. Stat. Soc. Ser C (applied statistics) **28**(1), 100–108 (1979)
5. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: An efficient k-means clustering algorithm: Analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 881–892 (2002)
6. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
7. Angiulli, F., Basta, S., Pizzuti, C.: Distance-based detection and prediction of outliers. IEEE transactions on knowledge and data engineering **18**(2), 145–160 (2005)
8. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high-dimensional datasets. Data Min. Knowl. Disc. **16**(3), 349–364 (2008)
9. Lu, W., Shen, Y., Chen, S., Ooi, B.C.: Efficient processing of k nearest neighbor joins using mapreduce. (2012) arXiv preprint arXiv:1207.0141
10. Ning, J., Chen, L., Zhou, C., Wen, Y.: Parameter k search strategy in outlier detection. Pattern Recogn. Lett. **112**, 56–62 (2018)

11. Lin, R.A.K.-l., Shim, H.S.S.K.: Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: Proceeding of the 21th International Conference on Very Large Data Bases, pp. 490–501. Citeseer (1995)
12. Zhang, Y., Wu, J., Wang, J., Xing, C.: A transformation-based framework for knn set similarity search. IEEE Trans. Knowl. Data Eng. **32**(3), 409–423 (2018)
13. Amorim, L.A., Freitas, M.F., da Silva, P.H., Martins, W.S.: A fast similarity search knn for textual datasets. In: 2018 Symposium on High Performance Computing Systems (WSCAD), pp. 229–236. IEEE (2018)
14. Samariya, D., Ma, J., Aryal, S., Zhao, X.: Detection and explanation of anomalies in healthcare data. Health. Inf. Sci. Syst. **11**(1), 20 (2023)
15. Ashour, A.S., Hawas, A.R., Guo, Y.: Comparative study of multiclass classification methods on light microscopic images for hepatic schistosomiasis fibrosis diagnosis. Health. Inf. Sci. Syst. **6**, 1–12 (2018)
16. Bajaj, V., Taran, S., Sengur, A.: Emotion classification using flexible analytic wavelet transform for electroencephalogram signals. Health. Inf. Sci. Syst. **6**, 1–7 (2018)
17. Chen, C., Zhu, Q., Wu, Y., Sun, R., Wang, X., Liu, X.: Efficient critical relationships identification in bipartite networks. World. Wide. Web. **25**(2), 741–761 (2022)
18. Rabie, A.H., Saleh, A.I.: A new diagnostic autism spectrum disorder (DASD) strategy using ensemble diagnosis methodology based on blood tests. Health. Inf. Sci. Syst. **11**(1), 36 (2023)
19. Tweets, M.: Twitter official blog [web-page]. 22 feb. Electronic resource https://blog.twitter.com/official/en_us/a/2010/measuring-tweets.html (2010)
20. Böhm, C., Krebs, F.: The k-nearest neighbour join: Turbo charging the kdd process. Knowl. Inf. Syst. **6**(6), 728–749 (2004)
21. Xia, C., Lu, H., Ooi, B.C., Hu, J.: Gorder: an efficient method for knn join processing. In: Proceedings of the Thirtieth International Conference on Very Large Data bases-Volume 30, pp. 756–767. (2004)
22. Yu, C., Cui, B., Wang, S., Su, J.: Efficient index-based knn join processing for high-dimensional data. Inf. Softw. Technol. **49**(4), 332–344 (2007)
23. Wang, J., Lin, L., Huang, T., Wang, J., He, Z.: Efficient k-nearest neighbor join algorithms for high dimensional sparse data. (2010) arXiv preprint arXiv:1011.2807
24. Zhang, C., Li, F., Jestes, J.: Efficient parallel knn joins for large data in mapreduce. In: Proceedings of the 15th International Conference on Extending Database Technology, pp. 38–49 (2012)
25. Yu, C., Zhang, R., Huang, Y., Xiong, H.: High-dimensional knn joins with incremental updates. Geoinformatica **14**(1), 55–82 (2010)
26. Yang, C., Yu, X., Liu, Y.: Continuous knn join processing for real-time recommendation. In: 2014 IEEE International Conference on Data Mining, pp. 640–649. IEEE (2014)
27. Ukey, N., Yang, Z., Zhang, G., Liu, B., Li, B., Zhang, W.: Efficient knn join over dynamic high-dimensional data. In: Australasian Database Conference, pp. 63–75. Springer (2022)
28. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., El Abbadi, A.: Vector approximation based indexing for non-uniform high dimensional data sets. In: Proceedings of the Ninth International Conference on Information and Knowledge Management, pp. 202–209 (2000)
29. Cui, B., Coi, B.C., Su, J., Tan, K.-L.: Indexing high-dimensional data for efficient in-memory similarity search. IEEE Trans. Knowl. Data Eng. **17**(3), 339–353 (2005)
30. Chakrabarti, K., Mehrotra, S.: Local dimensionality reduction: A new approach to indexing high dimensional spaces. Technical Report, TR-MARS-00-04, University of California at Irvin (2000). http://www-db.ics.uci.edu/pages/publications/
31. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y.: Efficiently processing snapshot and continuous reverse k nearest neighbors queries. The VLDB Journal **21**(5), 703–728 (2012)
32. Berchtold, S., Böhm, C., Kriegal, H.-P.: The pyramid-technique: Towards breaking the curse of dimensionality. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, pp. 142–153 (1998)
33. Yu, C., Ooi, B.C., Tan, K.-L., Jagadish, H.: Indexing the distance: An efficient method to knn processing. In: Vldb, vol. 1, pp. 421–430. (2001)
34. Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C., Zhang, R.: idistance: An adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. (TODS) **30**(2), 364–397 (2005)
35. Cui, B., Ooi, B.C., Su, J., Tan, K.-L.: Contorting high dimensional data for efficient main memory knn processing. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 479–490 (2003)
36. Weber, R., Schek, H.-J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB, vol. 98, pp. 194–205. (1998)
37. Ooi, B.C., Tan, K.-L., Yu, C., Bressan, S.: Indexing the edges-a simple and yet efficient approach to high-dimensional indexing. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 166–174. (2000)

38. Pillai, K.G., Sturlaugson, L., Banda, J.M., Angryk, R.A.: Extending high-dimensional indexing techniques pyramid and iminmax (θ): Lessons learned. In: Big Data: 29th British National Conference on Databases, BNCOD 2013, Oxford, UK, July 8-10, 2013. Proceedings 29, pp. 253–267. Springer (2013)
39. Gionis, A., Indyk, P., Motwani, R., et al.: Similarity search in high dimensions via hashing. In: Vldb, vol. 99, pp. 518–529 (1999)
40. Hu, Y., Yang, C., Zhan, P., Zhao, J., Li, Y., Li, X.: Efficient continuous knn join processing for real-time recommendation. Pers. Ubiquit. Comput. **25**(6), 1001–1011 (2021)
41. Böhm, C., Krebs, F.: Supporting kdd applications by the k-nearest neighbor join. In: International Conference on Database and Expert Systems Applications, pp. 504–516. Springer (2003)
42. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, pp. 47–57 (1984)
43. Ukey, N., Yang, Z., Li, B., Zhang, G., Hu, Y., Zhang, W.: Survey on exact knn queries over high-dimensional data space. Sensors **23**(2), 629 (2023)
44. Jiaqi, J., Chung, Y.: Research on k nearest neighbor join for big data. In: 2017 IEEE International Conference on Information and Automation (ICIA), pp. 1077–1081. IEEE (2017)
45. Souza, V., Carvalho, L.O., de Oliveira, D., Bedo, M., Santos, L.F.: Adding result diversification to k nn-based joins in a map-reduce framework. In: International Conference on Database and Expert Systems Applications, pp. 68–83. Springer (2023)
46. Nalepa, F., Batko, M., Zezula, P.: Speeding up continuous knn join by binary sketches. In: Advances in Data Mining. Applications and Theoretical Aspects: 18th Industrial Conference, ICDM 2018, New York, NY, USA, July 11-12, 2018, Proceedings 18, pp. 183–198. Springer (2018)
47. Shahvarani, A., Jacobsen, H.-A.: Distributed stream knn join. In: Proceedings of the 2021 International Conference on Management of Data, pp. 1597–1609 (2021)
48. Lee, H., Chang, J.-W., Chae, C.: knn-join query processing algorithm on mapreduce for large amounts of data. In: 2021 International Symposium on Electrical, Electronics and Information Engineering, pp. 538–544 (2021)
49. Allheeib, N., Adhinugraha, K., Taniar, D., Islam, Md. Saiful.: Computing reverse nearest neighbourhood on road maps. World. Wide. Web. 1–32 (2022)
50. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Vldb, vol. 97, pp. 426–435 (1997)
51. Chakrabarti, K., Mehrotra, S.: Local dimensionality reduction: A new approach to indexing high dimensional spaces. In: VLDB Conference (2000)
52. Achlioptas, D.: Database-friendly random projections. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 274–281 (2001)
53. Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., Equitz, W.: Efficient and effective querying by image content. Journal of intelligent information systems **3**, 231–262 (1994)
54. Leon, S.J., Björck, Å., Gander, W.: Gram-schmidt orthogonalization: 100 years and more. Numer. Linear Algebra Appl. **20**(3), 492–532 (2013)
55. Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., Zheng, Y.: Nus-wide: a real-world Web image database from national university of singapore. In: Proceedings of the ACM International Conference on Image and Video Retrieval, pp. 1–9. (2009)

Authors and Affiliations

Nimish Ukey[1] · Guangjian Zhang[1] · Zhengyi Yang[1] · Binghao Li[2] · Wei Li[3] · Wenjie Zhang[1]

Nimish Ukey
n.ukey@unsw.edu.au

Guangjian Zhang
guangjian.zhang@unsw.edu.au

Binghao Li
binghao.li@unsw.edu.au

Wei Li
wei.li@hrbeu.edu.cn

Wenjie Zhang
wenjie.zhang@unsw.edu.au

1 School of Computer Science and Engineering, The University of New South Wales, Sydney 2052, NSW, Australia

2 School of Minerals and Energy Resources, The University of New South Wales, Sydney 2052, NSW, Australia

3 College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, Heilongjiang, China