Journal of Big Data

SURVEY Open Access

Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications

Rajib Kumar Halder[1], Mohammed Nasir Uddin[1], Md. Ashraf Uddin[2*], Sunil Aryal[2] and Ansam Khraisat[2]

*Correspondence:
ashraf.uddin@deakin.edu.au

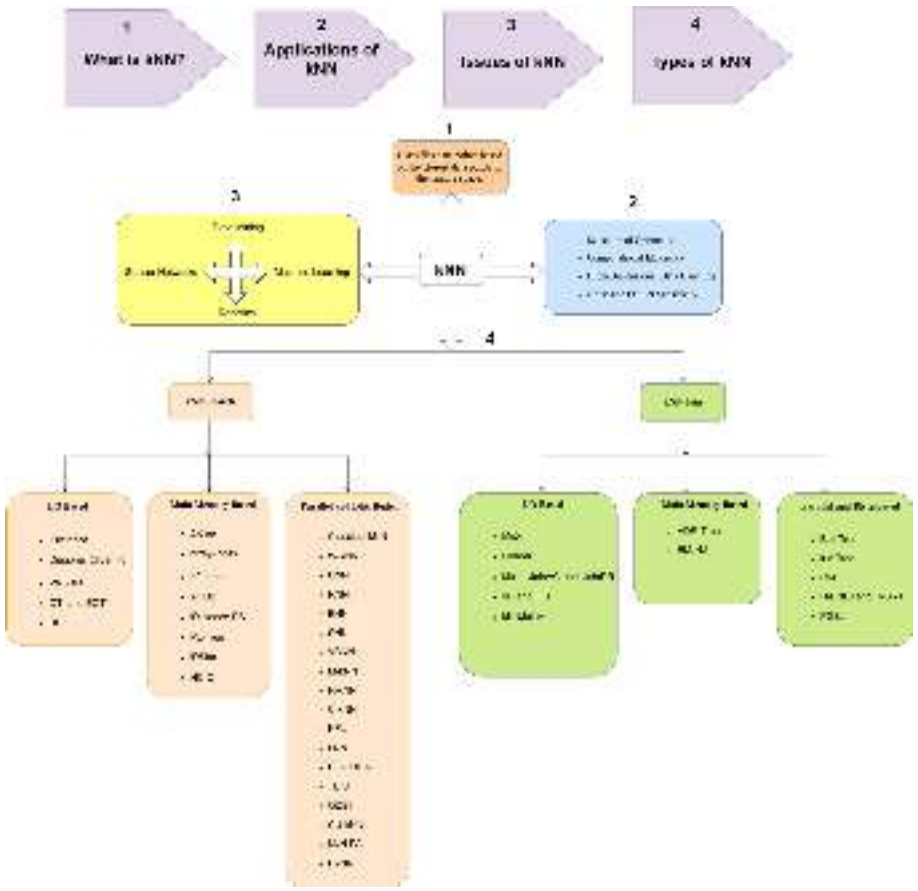
[1] Department of Computer Science and Engineering, Jagannath University, Dhaka 1100, Bangladesh
[2] Deakin Cyber Research and Innovation Centre, Deakin University, Geelong, Australia

Abstract

The k-Nearest Neighbors (kNN) method, established in 1951, has since evolved into a pivotal tool in data mining, recommendation systems, and Internet of Things (IoT), among other areas. This paper presents a comprehensive review and performance analysis of modifications made to enhance the exact kNN techniques, particularly focusing on kNN Search and kNN Join for high-dimensional data. We delve deep into 31 kNN search methods and 12 kNN join methods, providing a methodological overview and analytical insight into each, emphasizing their strengths, limitations, and applicability. An important feature of our study is the provision of the source code for each of the kNN methods discussed, fostering ease of experimentation and comparative analysis for readers. Motivated by the rising significance of kNN in high-dimensional spaces and a recognized gap in comprehensive surveys on exact kNN techniques, our work seeks to bridge this gap. Additionally, we outline existing challenges and present potential directions for future research in the domain of kNN techniques, offering a holistic guide that amalgamates, compares, and dissects existing methodologies in a coherent manner.

Keywords: K-nearest neighbors join, K-nearest neighbors search, Exact K-nearest neighbors, High dimensional data, Performance analysis

Graphical Abstract



Introduction

K-Nearest Neighbors (kNN) is a method in supervised machine learning, originally developed by Evelyn Fix and Joseph Hodges in 1951 and later refined by Thomas Cover [1]. This algorithm is extensively utilized across diverse fields such as data mining, recommendation systems, and the Internet of Things (IoT), playing a pivotal role in the advent of Industry 4.0. Specifically, in the realm of data mining, kNN is instrumental in classifying human activities, registering iterative closest points, and recognizing patterns. Additionally, it proves highly reliable in systems aimed at detecting intrusions and faults. These applications of kNN are thoroughly discussed in Sect. "Applications of kNN".

The K-Nearest Neighbors (kNN) algorithm operates as a non-parametric, instance-based learning method, commonly employed in supervised learning tasks, including classification and regression. Contrasting with model-based learning approaches that deduce a function from training data to make predictions, kNN is categorized as a lazy learning algorithm. It formulates predictions by analyzing the data structure in real-time upon the introduction of new instances, without necessitating a preceding explicit training phase. The K-Nearest Neighbors (kNN) algorithm operates on the principle of

likelihood of similarity. It posits that similar data points tend to cluster near each other in space. Consequently, the prediction for a new data instance is based on its proximity to existing instances in the training set.

Steps in the kNN algorithm:

- Select k: Begin by choosing the number of nearest neighbors to consult. This number, k, is a critical hyperparameter that you adjust based on your dataset's specific characteristics. The optimal value of k is essential for the accuracy of the algorithm's predictions. A smaller k value can make the algorithm sensitive to noise and overly flexible, whereas a larger k can render it computationally intensive and prone to underfitting.
- Calculate Distances: Compute the distance between the new instance and all points in the training dataset. Common metrics for this calculation include Euclidean, Manhattan, and Minkowski distances. The selection of a distance metric can significantly affect the algorithm's performance, particularly in relation to the dataset's characteristics.
- Identify Nearest Neighbors: Order all points in the training set from nearest to farthest from the new point, and select the closest k points.
- Aggregate Neighbor Responses: For classification tasks, the prediction is typically the majority label among these k nearest neighbors. For regression, it might be the average or median of the neighbors' values.

Choice of parameters and its implications:

- Selecting k: The choice of k has a profound impact on the model's behavior. A lower k can cause the model to overfit, capturing noise instead of representing the true underlying patterns of the data. On the other hand, a higher k tends to overly smooth the decision boundary, which can lead to underfitting.
- Distance Metric: While Euclidean distance is the most commonly utilized, other metrics like Manhattan or Minkowski might be more suitable in scenarios involving high-dimensional data or when different scaling or sensitivity to particular dimensions is required.

The recent discourse on K-nearest neighbors (kNN) algorithms has highlighted several critical issues:

1. Selection of Optimal k: Determining the most appropriate number of neighbors (k) remains a challenge as it significantly impacts the algorithm's accuracy and generalization ability.
2. Computational Efficiency: The classic kNN algorithm can be computationally intensive, particularly with large datasets, due to its need to compute distances between points for each query.
3. High-Dimensional Data Handling: kNN's performance can deteriorate in high-dimensional spaces due to the curse of dimensionality, where distances become less meaningful.
4. Noise and Outlier Sensitivity: The algorithm's reliance on the nearest neighbors makes it susceptible to noise and outliers in the data.

To address kNN algorithm challenges, scholars have innovated by creating adaptive algorithms to dynamically select the optimal k value, enhancing the algorithm's sensitivity to the specificities of the data. For computational efficiency, dimensionality reduction techniques have been applied to mitigate the curse of dimensionality. Advanced distance metrics and weighting schemes improve robustness against noise and outliers, enhancing the algorithm's accuracy in high-dimensional spaces. These adaptations, including variants like Adaptive kNN, Weight adjusted kNN, and Fuzzy kNN, are particularly pivotal in domains requiring high precision, such as healthcare diagnostics, showcasing a tailored approach to overcoming kNN's inherent limitations. We have noticed that a lot of current research focuses on faster, approximated methods like approximate nearest neighbor (ANN) algorithms [2–6] to handle complex data. These methods are fast but might not always provide the most accurate results. This implies that the nearest neighbors obtained might not accurately represent the actual k nearest neighbors. In contrast, we are particularly interested in the exact kNN methods that ensure the highest level of accuracy in finding the true closest data points. This paper is dedicated to discussing these exact kNN methods, given their critical importance in scenarios where precision is key. This paper focuses on two main types of kNN queries: kNN Search and kNN Join, exploring their important roles, details, and opportunities for improvement. kNN Search aims to find the 'K' nearest data points to a specific query point, essential for applications that need quick and accurate data retrieval. On the other hand, kNN Join finds the 'K' closest points for every query point in the dataset, helping to uncover hidden patterns and relationships in the data, leading to a deeper understanding of the data's structure and meaning. The research questions we aim to address are as follows:

RQ1: What are the applications and critical issues of kNN?
RQ2: What are the various state-of-the-art variants of kNN that have emerged over time?
RQ3: What are the distinct roles, and methodologies for both kNN Search and kNN Join queries, considering their respective importance in facilitating rapid data retrieval and uncovering hidden patterns and relationships within datasets?
RQ4: What are the strengths and weaknesses associated with various iterations of both kNN search and kNN join techniques?
RQ5: How do various R-tree variants, including R-tree, R*-tree, Hilbert R-tree, PR-trees, KD-tree, Ball-tree, VP tree, and MVP tree, contribute to the efficient processing of low-dimensional datasets in kNN queries, and what are their comparative advantages and limitations?
RQ6: How do parallelization, dimensionality reduction techniques, and partitioning methods address the challenges of efficient kNN query processing in high-dimensional spaces, and what are their comparative advantages and limitations in overcoming the "curse of dimensionality"?
RQ7: How do we address the challenges faced by traditional kNN methods, and how do we enhance information extraction, optimize computational efficiency, integrate ensemble learning, and improve classification accuracy for large-scale data classification tasks?

kNN in low-dimensional space: an overview

Basic kNN search approach To determine the k closest neighbors for a given query point, the foundational kNN Search method is utilized. This technique is frequently referred to as the brute force (BF) method or the exhaustive search approach. When applying this method, every data point in the dataset is scanned to identify the k nearest points based on the distances from the query point to all other data points. The primary drawback of this approach is its computational intensity. Calculating the Euclidean distance for a single kNN query comes at a cost of $O(nd)$, where "n" is the sample count and "d" signifies the dimensionality of the datasets. When the dataset is substantial, or numerous queries are pending, the query execution time can become prohibitively long.

kNN join and its advantages Böhm and Krebs were the first to introduce the concept of kNN Join [7, 8]. This study emerged from the realization that computing the nearest neighbors for all query points concurrently is significantly faster than doing so individually. The adoption of kNN Join has amplified the performance of various applications. Some notable applications include: k-means clustering [9, 10], Outlier detection [11, 12], kNN classification [13, 14], k distance diagrams, Missing value computation, and others [8]. Several research works delve into kNN queries specifically in low-dimensional space [15–22]. The evolution of R-tree variants reflects the ongoing effort to enhance the efficient processing of low-dimensional datasets, leading to the proposition of several versions. The original R-tree [15] divides the minimum bounding rectangle (MBR) to manage spatial data efficiently. The R*-tree [16] improves upon this by taking overlap into account when dividing an MBR, reducing overlap and coverage for better performance. The Hilbert R-tree [17] further optimizes the grouping of related MBRs based on Hilbert ordering, which improves query performance. PR-trees [18] handle large data volumes by using priority rectangles, which help in managing extensive datasets more effectively. Other notable structures include the KD-tree [19], which is a binary search tree method designed to expedite the neighbor search process by recursively partitioning the data space. The Ball-tree [20] uses a branch and bound approach to optimize distance computations, effectively managing high-dimensional data. The Vantage Point Tree (VP tree) [21] addresses complex search issues by organizing data points based on their distances to a selected vantage point, allowing efficient similarity searches. The Multi-Vantage Point (MVP) tree [22] builds on this by using multiple vantage points to divide space into spherical slices, providing a robust distance-based index structure for similarity searches. These innovations in tree structures and indexing methods significantly contribute to the advancement of efficient data retrieval processes in various dimensions.

kNN in high-dimensional space: an overview

The efficient processing of k-Nearest Neighbors (kNN) queries becomes increasingly complex with the rise in data dimensionality, often termed as the "curse of dimensionality" [23–25]. Traditional indexing algorithms like B-tree and R-tree are optimized for lower dimensions, but their performance diminishes rapidly as the dimensionality increases. This phenomenon necessitates innovative solutions to facilitate efficient and speedy kNN queries in high-dimensional spaces. The challenges of high-dimensional kNN queries are significant and multifaceted. Two primary challenges exist:

first, identifying the k nearest neighbors (kNN), and second, computing distances rapidly and efficiently. The complexity of identifying the kNN increases exponentially with the number of dimensions, making traditional methods less effective. Efficiently computing distances in high-dimensional spaces is equally challenging due to the curse of dimensionality, which can lead to a rapid increase in computational requirements and a decrease in performance. These challenges necessitate the development and implementation of advanced methodologies and optimizations to ensure efficient and accurate kNN query processing. These challenges are compounded by increased dimensions, necessitating the exploration of novel methodologies and optimizations [26–28]. The traditional indexing dilemma arises because B-tree and R-tree families, though proficient in low-dimensional settings, are adversely impacted by increased dimensions. Consequently, sequential linear scans often become the fastest retrieval method, albeit inefficient. Strategies for enhancing kNN query processing include parallelization, dimensionality reduction, and partitioning methods. Parallelization divides the computational task into segments, enabling concurrent processing on different units [29]. This approach utilizes clusters, GPUs, or multi-core machines to bolster processing speed without diminishing computational requirements, and researchers have demonstrated significant speed enhancements, particularly by exploiting GPUs for parallelized brute-force searches. Dimensionality reduction (DR) transforms high-dimensional data into a more manageable, lower-dimensional format [30–33]. Techniques such as Principal Component Analysis (PCA) are renowned for their efficiency and scalability, accelerating searches and data processing, and marking a cost-effective solution to the challenges posed by higher dimensions. Partitioning methods are classified into space-based and data-based categories. These techniques incorporate diverse tree structures like R-tree, R*-tree, Ball tree, and KD-tree, each designed for efficient data space segmentation and distance calculation pruning. Given the inadequacy of traditional tree structures in high-dimensional settings, innovative models like M-tree, Δ-tree, and HDR-tree have emerged [26, 34]. The objective of these adaptations is to optimize kNN query processing amidst the intricate landscape of high-dimensional data.

Survey scope

This article endeavors to conduct an in-depth examination of exact kNN querying techniques, notably kNN Search and kNN Join, within the context of high-dimensional spaces. This focus arises from the observation that a significant body of recent works is centered on exact kNN queries amidst high-dimensional datasets. Previous surveys have either concentrated on kNN techniques within low-dimensional spaces [35–37] or explored approximate methodologies in high-dimensional environments [38, 39]. Additionally, while there are MapReduce-based studies that incorporate distributed and parallel survey work [40, 41], a void exists in literature that holistically addresses exact kNN Join methodologies.

Objectives

We aim to fill aforementioned gap by giving a thorough review of the current specific kNN techniques, looking closely at how they work with complex data. We will organize these methods into clear categories and compare them side by side, highlighting their

unique features, how efficient they are, and where they can be best applied. In simpler terms, we are making a complete guide that breaks down and compares these complex methods in a way that is easy to understand.

Motivations

The imperative to undertake this survey is rooted in the following considerations:

1. The ubiquity and escalating popularity of kNN queries in high-dimensional spaces, courtesy of their expansive applicability across myriad domains in recent times.
2. While kNN Search has been the focal point of numerous scholarly contributions, a comprehensive survey synthesizing exact kNN techniques for high-dimensional data remains elusive.
3. The literature on kNN Join, although rich with comparative studies on diverse exact and approximate MapReduce-centric approaches [40, 41], lacks a dedicated survey spotlighting solely on exact kNN Join methodologies pertinent to high-dimensional realms.

Contributions

We are exploring advanced ways to use kNN Search and kNN Join for high-dimensional data. Here is a simplified breakdown of what we have covered:

1. Methodological Overview: We extensively explored exact kNN techniques for high-dimensional datasets, encompassing 31 search methods and 12 join methods.
2. Analytical Insight: Each method has been critically evaluated, detailing its strengths, limitations, the evaluation metrics used, and the datasets upon which they were tested.
3. Resource Provision: For practical engagement, the source code of the kNN methods has been provided, facilitating direct experimentation and comparative analysis for readers.
4. Research Horizons: We have identified existing challenges and outlined potential directions for future investigations in the realm of kNN techniques.

Structure of the article Sect. "Problem definition" lays out the foundational terminology and outlines the problem we address. In Sect. "Article selection process", we detail the criteria and process behind the selection of articles for our study. Sects. "kNN search" and "kNN join approach" dissect the mechanisms of kNN queries, specifically focusing on kNN Search and kNN Join techniques, through the lens of computing paradigm classifications. Sect. "Applications of kNN" discusses the practical applications of kNN. A comparative analysis of the methodologies discussed is presented in Sect. "Comparative analysis". Sect. "Discussion" contains various perspectives and viewpoints on the application of k-Nearest Neighbors from other scholars. The document concludes with Sect. "Conclusion", summarizing our findings, and Sect. 11, where we identify challenges and outline future research directions.

Problem definition

In this section, we provide clear definitions for key terms frequently referenced throughout this paper.

Definition 1 (kNN search)

Let D be a dataset containing n points in a d-dimensional space, q a query point in the same space, k the number of nearest neighbors to retrieve, dist(p, q) a function that calculates the distance between point p and query point q. The kNN search problem is to find a set R containing k points from D such that for each point $p \in R$, there is no other point $p' \in D$ where $dist(p, q) > dist(p', q)$. In other words, every point in R is among the k closest points in D to the query point q. Figure 1 shows the example of kNN Search approach for $k=3$. Example: Given $k=3$, the kNN Search method identifies the three closest points in dataset D to the query point s_1.

Definition 2 (kNN join)

Let D_1, D_2 be two datasets containing n points in a d-dimensional space, k the number of nearest neighbors to join, $dist(p_1, p_2)$ a function that calculates the distance between point $p_1 \in D_1$ and query point$p_2 \in D_2$. The kNN join problem is to find pairs of points (p_1, p_2) such that$p_1 \in D_1$, there are exactly k points $p_2 \in D_2$ with minimal distance$dist(p_1, p_2)$. It means for each point in D_1, we find its k closest points inD_2. Figure 2 shows the example of kNN Join approach for $k=2$. Example: Given $k=2$, the kNN Join method determines the two closest points from the D_2 dataset for every point within the D_1 dataset.

Definition 3 (distance range)

A distance range is characterized by identifying all pairs of elements from two distinct datasets, R and S, that fall within a specified distance threshold, θ. In this context, dataset R contains elements $\{r_1, r_2, r_3, \ldots., r_n\}$ and dataset S consists of elements $\{s_1, s_2, s_3, \ldots., s_m\}$. The pairings from the two sets are determined by the criteria $(r, s) : r \in R,\ s \in S,\ d(r, s) \leq \theta$, where d(r, s) denotes the distance between elements r, s, and θ represents the user-defined distance threshold. Every pair that satisfies this condition is included in the result.

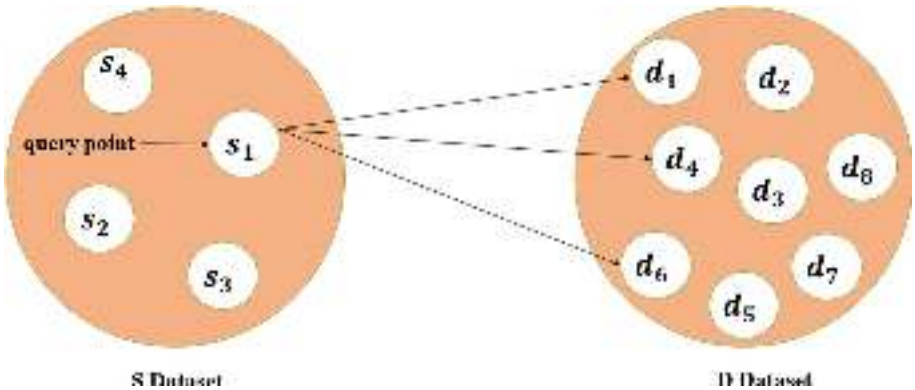


Fig. 1 Example of kNN Search approach for $k=3$

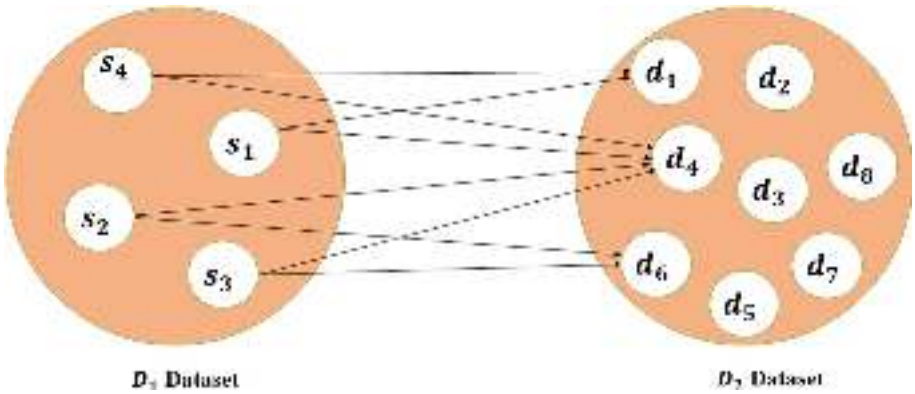


Fig. 2 Example of kNN Join approach for k = 2

Definition 4 (k-distance join)

The closest point query, equivalently termed as the k-Distance Join, is a method employed to ascertain the k most analogous pairings across two distinct datasets, R and S. Here R is defined as $R = \{r_1, r_2, r_3, \ldots., r_n\}$ and $S = \{s_1, s_2, s_3, \ldots., s_m\}$. The k -Distance Join aims to identify a subset KDJ(R, S) consisting of k pairs, each derived from the Cartesian product R × S, wherein each pair $(r_i, s_j) \in KDJ(R, S)$ satisfies the condition that the distance $d(r_i, s_j)$ is lesser or equal to the distance of any other pair (r_k, s_j) is lesser or equal to the distance of any other pair $(r_k, s_l) \in R \times S$. The range of k is bounded as $1 \leq k \leq \min(n, m)$, ensuring the derivation of the closest pairs without exceeding the total number of elements in either dataset.

Definition 5 (reverse kNN join)

Let R and S be two distinct datasets comprised of points situated within a d-dimensional space, denoted as R^d. Let the Euclidean distance function $d(r_i,s_j)$ measure the distance between any two given data points $r_i \in R$ and $s_j \in S$. Additionally, consider a natural number $k \in N^+$. Under these conditions, the outcome of a reverse kNN Join in correlation to a query data point s_j is defined as a subset $R_{kNN}(R, s_j) \subseteq R$, which incorporates data points that count s_j as one of their k nearest neighbors. Mathematically, it can be articulated as: $R_{kNN}(R, s_j, k) = \{r_1, r_2, r_3, \ldots., r_n\} \subseteq R$.

Definition 6 (dynamic kNN join)

R And S as two datasets consisting of points situated within a d-dimensional space R^d. The Euclidean distance between two points $r_i \in R$ and $s_j \in S$ is computed using the distance function $d(r_i, s_j)$. Let k be a positive natural number such that $k \in N^+$ and $1 \leq k \leq |s|$. In this context, a dynamic kNN Join refers to the process of dynamically identifying and associating similar data points across datasets R and S based on their Euclidean distance. The resultant set from this join operation is denoted asDkNNJ(R, S, k) ⊆ R × S, and it contains pairs of points (r_i, s_j) such that for each $r_i \in R$, s_j is among its k closest neighbors inS. The formal definition is: $DkNNJ(R, S, k) = \{(r_i, s_j) | \forall r_i \in R, s_j \notin kNN(S, r_i, k\}$. Furthermore, the dynamic nature of this join implies a real-time adaptation to the modifications in datasetS. For every

insertion or deletion operation involving an item $s_i \in S$, the affected set of users $r_a : \text{RkNN}(s_i)$ where $s_i \in S$ and $\text{RkNN}(s_i) \subset R$, is identified. Subsequently, updates are implemented to reflect the changes in the join results, ensuring that each point in r_a maintains its k closest neighbors in the modified datasetS. The updated join is expressed as:$\text{kNN}(S, r_a, k) \subseteq R \times S$.

Definition 7 (approximate nearest neighbour)

The Approximate Nearest Neighbor (ANN) search quickly identifies points in a dataset that are close to a query point, but not necessarily the closest. This approach is beneficial in high-dimensional spaces where exact searches are costly. In recommendation systems, for instance, it is often sufficient to suggest items that are "close enough" rather than pinpointing the most similar one. This can lead to unexpected and interesting discoveries for users. Similarly, in kNN classification, points that are near each other are often of the same class, even if they are not the absolute closest.

Article selection process

Search strategy and data sources

In 2022, we conducted a systematic review by searching various scientific databases, including Scopus,,ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink, Embase, Web of Science, PubMed, Research, Gate, Wikipedia, and others as listed in Table 1. The review was limited to studies conducted in English. We ran the search using a combination of restricted vocabularies (MESH words) and free-text terms to search electronic databases. Using the Google search engine, they also used relevant keywords to perform a general search for gray literature, such as conference papers, research projects, theses, and dissertations. The PRISMA flowchart conducted all of these procedures.

Table 1 Search strategy and keywords

Database	Scopus,, ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink, Embase, Web of Science, PubMed, Research, Gate, Wikipedia
Limits	Language (only resources with English), Species (studies on K Nearest Neighbor)
#1	"k Nearest Neighbor (kNN) " OR "kNN", OR "Variants of kNN", OR "Modified kNN "
#2	("Weighted kNN" OR "Condensed Nearest Neighbor" OR "Reduced Nearest Neighbor" OR "Edited Nearest Neighbor" OR "Edited Nearest Neighbor" OR "Selective Nearest Neighbor" OR "Voronoi Boundary Nearest Neighbor" OR "Model Based k Nearest Neighbor" OR "Ranked Based k Nearest Neighbor" OR OR "Clustered k Nearest Neighbor" OR "Ball Tree k Nearest Neighbor" OR "k-d Tree Nearest Neighbor" OR "Nearest Future Line Neighbor" OR "Local Nearest Neighbor" OR "Principle Axis Tree Nearest Neighbor"OR "iDistance" OR "Diagonal Ordering" OR "*VA*$^{+}$-file" OR "OTI and EOTI" OR "BP" OR "Δ-tree" OR "array-index" OR "Δ^{+}-tree" OR "ACDB" OR "iDistance-PS" OR "PL-Tree" OR "iDStar" OR "HC–O" OR "BF-CUDA" OR "CUBLAS" OR "TBiS" OR "QDBI" OR "CU-kNN" OR "kNN-PA" OR "HkNN" OR "MuX" OR "Gorder" OR "iJoin" OR "iJoinAC" OR "iJoinDR" OR "IIB" OR "IIIB algorithm" OR "kNNJoin +" OR "HDR-Tree" OR "CTD-kNNJ" OR "EkNNJ" OR "H-BNLJ" OR "H-BRJ" OR "PGBJ"
#3	#1 AND #2

Data selection (inclusion and exclusion criteria)

To identify relevant studies, this investigation comprehensively searched conference papers and journal articles from specific databases without any time restrictions. However, the researchers excluded other types of research, such as letters to the editor, educational reports, brief communications, editorial commentary, editorials, protocols, standards, and industry papers. Duplicate and irrelevant studies, as well as publications without full-text access, were also eliminated. The study focused on modified kNN applications across various sectors and included only articles written in English. Moreover, articles discussing data mining and machine learning, as well as those that were not related to the study, were excluded. We have selected 43 (Year: 1967 to 2022) papers that met the inclusion criteria from the initial pool of 1056 studies. Figure 3 depicts how we set the article after the PRISMA declaration.

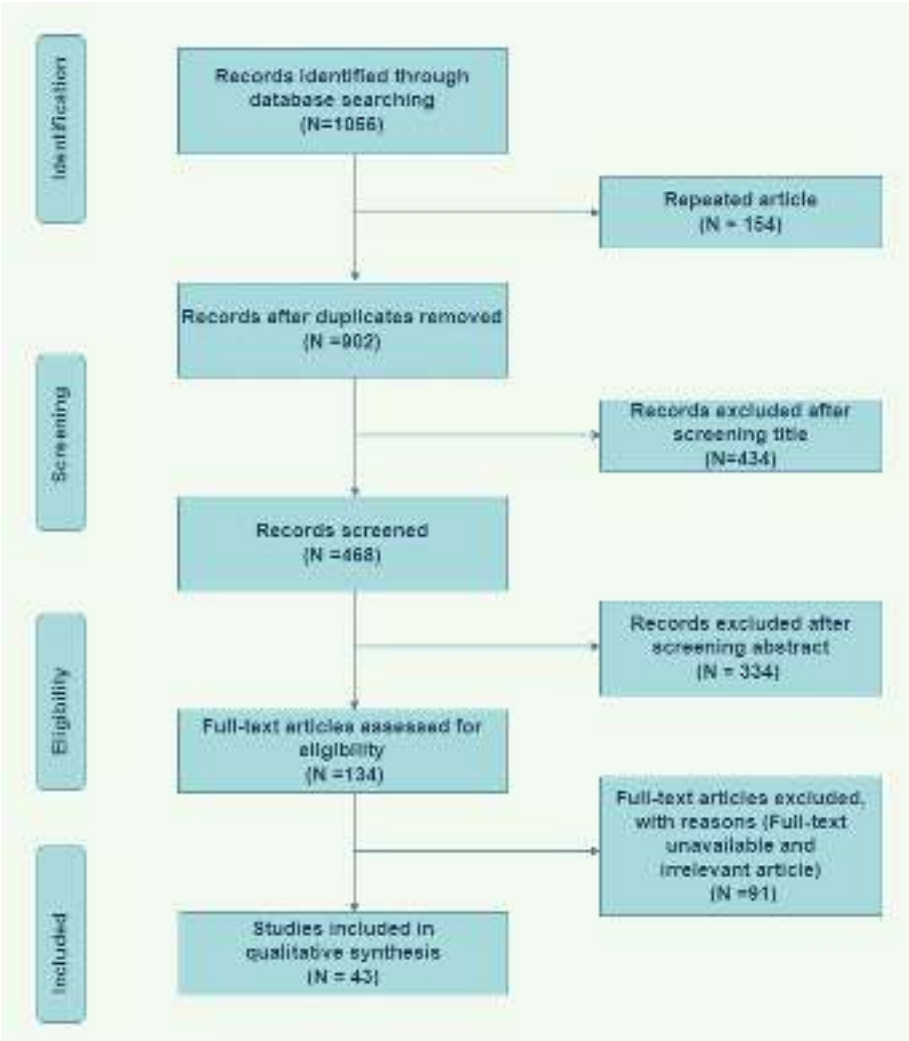


Fig. 3 Flow diagram of the article selection process from 1967 to 2022

Data extraction

We have utilized a form that divided the extracted information from the selected studies into three categories: general information, objective and method, and significant results. The general information category included the publication's title, author, date, and location. The objective and method types included information on the research objectives, data analysis, data mining methodology, and resource data used in the study. Lastly, the significant results category captured the three main conclusions drawn from the research. To facilitate a comprehensive grasp of the methodologies and their evolution, a chronological representation of the kNN Join and kNN Search techniques is delineated in Fig. 4. The upper segment of the figure enumerates all the kNN Join techniques, while the kNN Search methodologies are cataloged in the lower section.

Furthermore, for an organized understanding, kNN Search and kNN Joins have been categorized based on distinct Computing Paradigms—I/O-based, main-memory-based, and parallel or distributed, as documented in Tables 2 and 3, respectively.

kNN search

In a significant number of database applications, the task of identifying an item similar to a specified query point is often complex and resource-intensive. Under these conditions, the kNN query proves to be especially beneficial. Numerous studies have been undertaken to solve the exact kNN issue, yet a considerable portion has been devoted to approximation methods. That said, there is also a segment of research focused on precise kNN solutions, with proposals such as HDR-tree and iDistance emerging to optimize the efficiency of kNN identification and minimize the associated computational expenses. This survey is primarily concentrated on these precise methodologies. An exhaustive exploration of an array of high-dimensional kNN Search and kNN Join methods is presented. To simplify the comprehension process, these methods have been systematically categorized. In this section, we focus on the diverse exact kNN Search

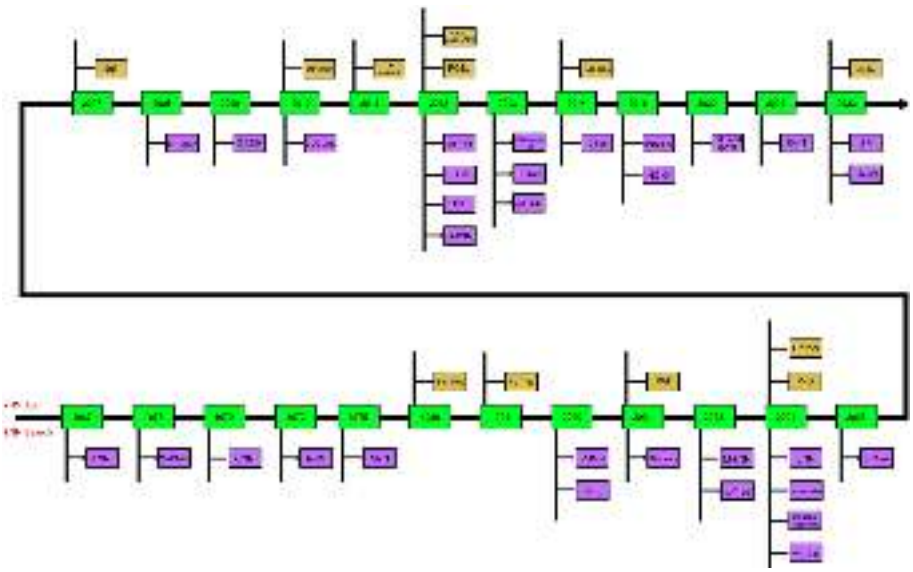


Fig. 4 Chronological representation of the kNN Join and kNN Search

Table 2 Classification of kNN Search approach based on Computing Paradigms

Categories	Algorithm	Time complexity
I/O based	iDistance [42, 43]	$O(logn)$
	Diagonal Ordering [44]	$O(logn)$
	VA^{+}-file [47]	$O(logn)$
	OTI and EOTI [48]	$O(n)$
	BP [52]	$O(nlogn)$
Main memory based	Δ-tree [26]	$O(n)$
	array-index [53]	$O(logn)$
	Δ^{+}-tree [26, 54]	$O(n)$
	ACDB [55]	$O(nlogn)$
	iDistance-PS [56]	$O(n)$
	PL-Tree [62]	$O(logn)$
	iDStar [63]	$O(n)$
	HC–O [66]	$O(n)$
Parallel and distributed	Classical kNN [69]	$O(n)$
	W-kNN [70]	$O(n)$
	CNN [71]	$O(n)$
	RNN [72]	$O(n)$
	ENN [73]	$O(n)$
	SNN [74]	$O(n)$
	VBNN [75]	$O(n)toO(logn)$
	M-kNN [76]	$O(n)$
	R-kNN [77]	$O(n)$
	C-kNN [78]	$O(n)toO(logn)$
	NFL [79]	$O(n)$
	LNN [80]	$O(n)$
	BF-CUDA [29]	$O(n)$
	TBiS [83]	$O(log^2 n)$
	QDBI [87]	$O(logn)$
	CU-kNN [88]	$O(n)$
	kNN-PA [89]	$O(nlogn)$
	HkNN [92]	$O(n)$

Table 3 Classification of kNN Join approach based on Computing Paradigms

Categories	Algorithm	Time complexity
I/O based	MuX [7, 8]	$O(nlogn)$
	Gorder [94]	$O(n^2)$
	iJoin, iJoinAC and iJoinDR [95]	$O(nlogn)$
	IIB and IIIB algorithm [96]	$O(nlogn)$
	kNNJoin + [28]	$O(nlogn)$
Main memory based	HDR-Tree [34]	$O(nlogn)$
	EkNNJ [98]	$O(nlogn)$
Parallel and distributed	Ball Tree [100]	$O(nlogn)$
	k-d Tree [19]	$O(nlogn)$
	PAT [101]	$O(logn)toO(n)$
	H-BNLJ and H-BRJ [102]	$O(n^2)$
	PGBJ [12]	$O(nlogn)$

methods, categorizing them based on different Computing Paradigms including I/O-based, main-memory-based, and parallel or distributed approaches.

I/O-based

These algorithms are optimized to minimize disk access and are especially beneficial when dealing with large datasets that cannot fit entirely into the main memory.

iDistance iDistance is an innovative index structure developed by researchers [42, 43] to improve kNN Search for high-dimensional (HD) data. In simple terms, it turns complex data into a more straightforward, one-dimensional format. The transformation involves:

1. Dividing the data into several sections.
2. Assigning a unique 'reference point' to each section.
3. Changing each section based on how similar data points are to their assigned reference point.

   For this, they used a $B+$-tree structure for fast access to the one-dimensional points and an array to store the reference points and their associated data. When searching, they initially look for the closest data points and expand the search area slowly until the desired neighbors are found. The efficiency of iDistance makes it stand out, even though most search methods become less effective with more data dimensions.

Diagonal ordering In a research study [44], a strategy called 'diagonal ordering' was introduced. It is a way to simplify high-dimensional data into a more manageable 1D format by slicing clusters of data diagonally. To help with data organization, they use a $B+$-tree structure. This method is somewhat similar to other approaches like iDistance and the Pyramid Technique. Here is the simple breakdown:

1. The vast data space is split into clusters.
2. Inside each cluster, vectors are organized in a diagonal order.
3. This organization helps turn complex vectors into one-dimensional values, which are then indexed using a B$+$-tree.

One significant benefit of this method is that it can quickly estimate the shortest distance between data points using the diagonal order. This helps improve search processes by quickly eliminating irrelevant data points without needing detailed calculations. The method uses an iterative approach to find the nearest neighbors: it begins searching with a small radius and expands until the desired neighbors are identified. To make data ordering effective, they applied PCA [45], prioritizing key characteristics, and used a clustering approach similar to iDistance's k-means clustering. Overall, the performance of the 'diagonal ordering' method was found to surpass several other methods, including X-tree [46], iDistance [42], and VA file [23].

VA^+-file Researchers in [47] introduced an innovative search method tailored for large, complex datasets. The VA^+-file technique is grounded in the scalar quantisation of data, proving especially efficient for kNN searches within non-uniform high-dimensional datasets. This approach enhances search performance by utilizing approximation techniques. The study presents a broad framework for approximate

kNN, explores various strategies for handling similarity queries, and introduces a new metric for evaluating these strategies. Additionally, a new method rooted in clustering is unveiled, amalgamating the strengths of different approaches to optimize progressive similarity searches.

OTI and EOTI In research documented in [48], a new approach to kNN searching, named the Optimum Triangle-Inequality (OTI), is proposed. This method reduces redundant distance calculations, making it more efficient than the traditional TI technique. However, OTI has a downside of requiring large space. To address this, the Enhanced Optimum Triangle-Inequality (EOTI) is introduced, optimizing the space and complexity associated with OTI. The KMC-TI-FS (TI) technique [49] divides the search process into two steps. Initially, data items are grouped using the k-means algorithm in an offline clustering phase. The second phase involves an online search where the TI approach identifies potential k nearest neighbors for a given query. This method accelerates the kNN search as distances between all items and the cluster center are pre-calculated and stored during the offline phase, eliminating the need for additional calculations. However, TI's limitation is that it cannot eliminate items in the marginal regions of adjacent clusters. OTI [48] was introduced to optimize this process by selecting an ideal cluster center that forms an appropriate triangle, improving the elimination process. However, OTI's high space complexity and increased calculation time needed refinement. That is where EOTI comes in, as an efficient solution balancing search performance with space and time complexity. EOTI records just two distances for each item, reducing space complexity to $O(N \times C)$. It registers the distances to the closest and farthest cluster centers, enabling efficient kNN searches with minimized space and time requirements.

BP Bregman distances are a common tool in fields like voice recognition and machine learning, including kNN Searches. They are used to process high-dimensional data, like that from multimedia systems. However, past methods using Bregman distances [50, 51] were limited to moderate dimensional data and struggled with efficiency in high-dimensional spaces due to challenges like cluster overlap and intensive computations. This issue is tackled in a new study [52], where researchers introduce a partition-filter-refinement structure for efficient high-dimensional kNN Search. The method involves dividing the high-dimensional space into smaller subspaces, running range queries, and then filtering the results. Innovations in this study include the use of the Cauchy inequality to calculate upper limits for searches within subspaces and introducing a Pearson Correlation Coefficient-based Partition (PCCP) to organize correlated dimensions into different subspaces. They also implemented Bregman Ball trees (BB-trees) [50] to accelerate searches within these low-dimensional subspaces and developed an integrated, disk-resident BB-forest index structure. During a search, query items are converted into a triple [52], and a range query is performed. Results are then filtered to identify the kNN. To optimize both efficiency and accuracy, the precise solution is modified into an Approximate BrePartition (ABP), marking the first non-metric method that utilizes Bregman distance and offers enhanced performance in high-dimensional spaces for kNN searches.

Main-memory-based

These are typically used for smaller datasets that can be loaded entirely into the memory, allowing for faster data access and processing.

Δ-tree A study [26] introduced the Δ-tree as a new index structure designed to optimize high-dimensional queries in main memory. The Δ-tree features a multilayered structure where each level, facilitated by PCA, presents increased dimensionality from the root to the leaves. Each layer aids in narrowing down the search area, thanks to the reduced dimensions that quicken distance calculations and optimize cache line size utilization. The Δ-tree is highly efficient in pruning the search region. Owing to the PCA property, distances between points in lower dimensions are always smaller than in higher ones. Thus, if the distance between a data point and the query in low dimension exceeds the original distance of the existing *k-th* nearest neighbor, that point is eliminated. However, the tree's effectiveness is influenced by a few factors:

1. It performs best with globally correlated datasets.
2. It requires processing the entire dataset to determine the PCA eigen matrix.
3. To maintain optimal performance, the entire tree needs frequent rebuilding.
4. Despite these challenges, with the correct number of levels and dimensions at each, the Δ-tree stands as a promising tool for efficient high-dimensional data queries.

array-index A study [53] introduced the array-index method to optimize the kNN Search performance for Data Partitioning Approaches, particularly on real, highly skewed and correlated datasets, while preserving their original features. The technique involves reading the data partitions formed by high-dimensional data partitioning approaches and converting these partitions into a linear form using ordering. This linearization is achieved by sorting the partitions based on the distance between a selected reference point and each partition's representative vector. The distances are then utilized to map the partitions into a 1D array-index space. This organization brings related partitions closer together, facilitating a more efficient search for kNN answer points. With this configuration, the algorithm needs to scan only a small region for any given query, leading to a notable reduction in kNN Search time when the array-index is incorporated into a Data Partitioning Approach.

Δ^+*-tree* A refined version of the Δ-tree, named the Δ^+-tree [26, 54], has been developed to tackle the original model's shortcomings. The foundational idea of the Δ^+-tree was first explored in [26], where the authors detailed the index approach and dynamic update techniques. The Δ^+-tree improves upon its predecessor by employing a more global approach to data splitting. The entire data space is divided into several clusters, with PCA being applied individually to each cluster, mitigating the first limitation of the Δ-tree. To address the second limitation, each cluster is further divided into smaller segments based on their distance from the center. This refined segmentation reduces the number of areas requiring evaluation during the search process. A Δ-tree is then constructed for every one of these smaller segments. This not only cuts down the computational cost but also reduces cache misses, making the Δ^+-tree a more efficient and effective tool for handling high-dimensional queries.

ACDB Hong et al. introduced an advanced kNN Search method, ACDB [55], optimized for high-dimensional indexing. It enhances search efficiency by reducing CPU usage through the application of the triangle inequality. The method incorporates two primary algorithms: the kNN Search algorithm and the Voronoi clusters' generation. Initially, the dataset is segmented into multiple Voronoi clusters, divided by hyperplanes. The distance between each cluster and its respective hyperplanes is calculated and recorded in a file. Items within each cluster are then indexed using the Euclidean distance measure. During each kNN query, the method dynamically establishes the lower distance limits for each cluster, arranging them in ascending order based on these limits. The search begins in the first cluster, seeking the kNN. Using the kNN distance, it verifies if this distance exceeds the lower limit of the succeeding cluster. If the kNN distance is less, the search concludes; otherwise, it proceeds to the next cluster, following the same procedure. The incorporation of the triangle inequality aids in accelerating the identification of the nearest items within a cluster, resulting in a reduced CPU load and enhanced search speed.

iDistance-PS A comprehensive evaluation of various partitioning techniques for the iDistance method was first presented in [56]. The researchers highlighted the substantial influence of these techniques on iDistance's performance and explored its utilization in contemporary applications and comparative studies. Since its introduction in [42, 43], iDistance has emerged as a leading high-dimensional indexing approach, notable for its effectiveness. Its application has expanded to challenging domains, including image retrieval [57] and video indexing [58], among others [59–61].

PL-tree The PL-tree [62] is an innovative indexing technique designed to enhance the efficiency of point queries, range queries, and kNN queries. This method systematically breaks down the original data space into hypercubes, each containing a specific number of data points. Every hypercube and the data points within are assigned a unique label through the Cantor pair function, ensuring that points within the same hypercube share the same label. The computational efficiency and bijective nature of the Cantor function enable a straightforward mapping of high-dimensional vectors to scalar labels. Whenever a subspace exceeds its data object limit, the partitioning and labeling process is triggered to divide the subspace further, ensuring an organized and efficient data retrieval system.

iDStar The study [63] examines several significant and manageable factors aiming to enhance the performance of kNN Search queries with the iDistance and iDStar algorithms. The authors also explore the difficulties associated with indexing in high-dimensional and tightly-clustered data spaces. Experimental results revealed that in spaces with fewer than 256 dimensions, the iDStar's approach of local division consistently outperforms the iDistance method, especially in clustered spaces. This study builds upon previous assessments and extensions of iDistance partitioning techniques and iDStar [56, 64, 65].

HC–O The authors introduced a novel caching mechanism [66] aimed at speeding up the item filtering stage of kNN Search, but faced two primary challenges: determining the optimal data point encoding strategy and deciding the number of bits required for encoding each data point. To address the first issue, they formulated and solved a new histogram optimization problem, and for the second, a cost model was developed to

automatically adjust the optimal bit count for encoding. This methodology is applicable to both exact tree-based and approximate LSH indexing methods. The process involves running queries, tallying the frequency of leaf node access, and prioritizing nodes for caching based on this data. A histogram is then constructed using an efficient method [113], which calculates the approximate data point representations contained within the leaf nodes. This caching approach can be adapted for any tree-based kNN Search algorithm with minor modifications and has proven to be more efficient than exact caching methods like iDistance, VP-tree [67], and VA-file [68].

Parallel or distributed

These are typically used for smaller datasets that can be loaded entirely into the memory. These are designed to split the workload across multiple processors or machines to handle large-scale datasets and computations efficiently.ng for faster data access and processing. Parallel data processing has become increasingly popular due to its effectiveness in enhancing the performance of various applications. The advent of General Purpose Graphics Processing Units (GPGPUs) has broadened the horizons for parallel processing. Utilizing NVIDIA's CUDA (Compute Unified Device Architecture) API has shown significant improvements in processing speed, offering a robust platform for executing parallel operations efficiently. This technology harnesses the power of graphics processing units (GPUs) to accelerate the performance of computing tasks, making it a sought-after solution in the realm of high-performance computing.

Classical kNN This [69, 147, 148] widely-used instance-based learning method is prominent in regression and classification applications. It is a non-parametric approach where all existing examples are stored and new ones are classified based on a similarity measure, often Euclidean distance. The process begins with data collection and preprocessing, where the dataset containing the features (input variables) and labels (output variables) is prepared. The next step is to choose the number of nearest neighbors (k) to consider when making a prediction. This is a crucial hyperparameter that can be tuned based on the performance of the model. Once k is selected, the algorithm calculates the distance between the new data point (query instance) and all the points in the training dataset. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance. After calculating these distances, the algorithm sorts them and selects the k smallest distances, which correspond to the k nearest neighbors. For classification tasks, the algorithm determines the majority class among the k nearest neighbors and assigns this class to the new data point. For example, if $k = 3$ and among the 3 nearest neighbors, 2 belong to class A and 1 belongs to class B, the new data point is classified as class A. For regression tasks, the algorithm computes the average (or weighted average) of the continuous values of the k nearest neighbors and uses this average as the predicted value for the new data point.

W-kNN The Weighted k Nearest Neighbor (W-kNN) [70] algorithm is an enhanced version of the traditional kNN algorithm. It incorporates the distance between the test and training instances to assign variable weights to each neighbor. Unlike the standard kNN, which treats all nearby neighbors with equal significance, W-kNN allocates higher weight to closer neighbors and lesser to those farther away. This algorithm calculates the distance, assigns weights according to the distances, and then selects the k

closest instances. The class of the test instance is determined by the weighted majority class among its k closest neighbors. The selection of the weight function is vital as it can significantly impact the performance of the W-kNN algorithm. Some weight functions include inverse distance weighting.

CNN The Condensed Nearest Neighbor (CNN) algorithm [71] is a machine learning classification method aimed at downsizing the training dataset to a representative subset. The algorithm starts with an empty set of selected instances (S) and adds the first instance from the dataset to it. It then examines each subsequent instance to see if it is classified correctly by the current set S. Instances that are classified accurately are removed, while those that are not are added to S. This continues until no more instances can be added to S. The reduced set S then serves as the training dataset for classifiers like kNN or decision trees. CNN is premised on the idea that a few representative instances can effectively classify the entire dataset, leading to enhanced efficiency and precision by reducing data redundancy.

RNN The Reduced Nearest Neighbor (RNN) algorithm [72] is a machine learning method designed for both classification and clustering. Conventional distance metrics, like the Euclidean distance, may fall short with high-dimensional datasets. RNN tackles this by identifying nearest neighbors for each data point, then selecting a "reduced neighborhood set" to minimize dataset density. This refining process continues, reducing neighbors in each step, until a stopping condition is met. The final reduced set can be used for classification or clustering tasks. While RNN is computationally efficient and handles noisy or sparse data well, its performance can be influenced by a chosen calibration parameter. In essence, it identifies neighbors using standard distances, calculates dataset density, selects subsets to decrease density, and repeats until a desired condition is achieved.

ENN The Edited Nearest Neighbor (ENN) algorithm [73] is a specialized tool in supervised machine learning, aimed at optimizing the training dataset by eliminating misclassified or irrelevant instances. The user specifies a parameter 'k' to identify the k nearest neighbors of each instance, utilizing a distance metric, often Euclidean. Instances are removed if the majority of their k neighbors belong to a different class, ensuring that the cleaned dataset, free from such discrepancies, fosters a more effective training of machine learning models. ENN acts as a preprocessing step to augment the dataset's integrity and quality.

SNN The Selective Nearest Neighbor (SNN) algorithm [74] excels in clustering high-dimensional data through a graph-based approach that categorizes data elements by similarity. It employs a distance metric, such as Euclidean distance or cosine similarity, to pinpoint each data point's k nearest neighbors. A graph is constructed with data points as vertices, connected to their k neighbors by edges. The similarity criterion determines edge creation—if two data points exhibit a similarity surpassing a set threshold, they are connected. The SNN clustering algorithm then takes over, allocating data points to clusters and merging clusters iteratively, guided by member similarity and a predefined threshold. Clusters amalgamate if a significant proportion of their members are mutual nearest neighbors with a surpassing similarity. The process concludes when no further cluster integration is feasible.

VBNN The Voronoi Boundary Nearest Neighbor (VBNN) algorithm [75] is a tool for clustering and classification tasks. It segments a multidimensional data space into distinct regions, utilizing Voronoi diagrams and neighboring data points. Each point is allocated a Voronoi cell based on its proximity to adjacent points. New data points are introduced and classified by measuring their distance to points on the Voronoi boundaries, and then assigning them to the nearest Voronoi region. Classification is further refined by calculating the Euclidean distance to all points, identifying the k nearest neighbors (with k being user-defined), and classifying the point according to the majority class of these neighbors within its assigned Voronoi region.

M-kNN Model Based kNN (M-kNN) Algorithm enhanced version of the conventional kNN algorithm [76] adopts a model-based approach for classification, optimizing the selection of local neighborhoods around each data point that consists mainly of similar class labels. Through the Euclidean distance measure, global neighborhoods are established in each iteration based on these local conglomerates. Each local area is represented by a central data point, selected for its encompassing information about the local region's size and similarity metrics. These representatives, each associated with an optimally distinct k value determined by the dataset, negate the need for user input and decrease the data points needed for classification, boosting efficiency. If a representative covers a new data point, the class label of that representative is assigned to the point. Uncovered data points undergo repeated procedures for classification, aligning with the kNN principle.

R-kNN Ranked Based KNN (R-kNN) Algorithm variant [77] integrates a ranking model to discern the most reliable neighbors, enhancing the weighted kNN approach. The algorithm employs a trained ranking model that assesses the reliability of adjacent labels in proximity to the true label set. For a new test instance, it identifies the k nearest neighbors through the standard kNN method, then reevaluates and re-ranks these neighbors based on their label similarity to the test instance using the ranking model. The final prediction employs a weighted voting mechanism, where weights are optimized to minimize a specific loss function, with performance assessed via the Hamming loss. Essentially, it is a refined similarity function learning from label distances, adaptable to various distance metrics for enhanced accuracy. The core goal is augmenting prediction accuracy by integrating a ranking model that assigns weights to neighbors' votes, reflecting their reliability.

C-kNN The Clustered k Nearest Neighbor (C-kNN) [78] algorithm addresses the challenges posed by multi-peaked data distributions in training datasets. Initially, it mitigates the multi-peak effect by removing samples near the boundaries of the training set. The remaining samples are then clustered by category using the k-means algorithm, and the cluster centers become the new training samples. This step streamlines the complexity and enhances the algorithm's performance. Each training sample is weighted according to the size of its cluster, favoring larger clusters for improved accuracy. The revised dataset facilitates the execution of the kNN algorithm, which classifies test samples based on the majority class label of their k nearest neighbors, making C-kNN a potent tool for tackling complex, multi-peaked data distributions.

NFL The Nearest Future Line (NFL) algorithm [79] is designed for image classification and retrieval, capitalizing on the idea of using multiple prototypes to encapsulate feature

diversity within a class. It conceptualizes each image as a point in feature space and forms a trajectory connecting the feature points of evolving prototype images. These trajectories collectively create a subspace representing the class. Unlike the nearest neighbor (NN) search algorithm, which overlooks the proximity of comparable images to this subspace, the NFL algorithm integrates this spatial relationship, enhancing the precision and efficiency of image classification and retrieval processes.

LNN The Local Nearest Neighbor (LNN) algorithm [80] introduces the concepts of the nearest neighbor line (NNL) and nearest neighbor plane (NNP) as pattern classifiers. They operate on the principle of local proximity, calculating the feature line or plane only for the nearest neighbors of a query point rather than the entire dataset. In a scenario with c classes, for a given query sample x, the NNL in the ith class is defined by the line connecting x's two closest neighbors in that class. The NL distance is the Euclidean distance from x to its projection on the NNL. The NNL with the minimum distance across all c classes is selected. For the NNP, it is constructed using three nearest neighbors in each class, and the one with the shortest distance to x is chosen. The LNN approach reduces computational expense compared to more complex classifiers like NFL or NFP.

BF-CUDA The process of identifying k-nearest neighbors (kNN) in extensive datasets of d-dimensional vectors is resource-intensive. One mitigation strategy is the organization of data through structures like binary trees. A study [29] explores this by implementing the brute force (BF) method of kNN search using NVIDIA's CUDA API. The BF method is essentially a two-step process, involving the calculation of distances between data points and then sorting them. For a specific query point q, the method:

1. Computes the distances between q and all other data points.
2. Sorts these distances in ascending order.
3. Identifies the k-nearest points.
4. Repeats the process for every query point.

In the experiments conducted in the study, a variant of the insertion sort proved more efficient than the comb sort for smaller k values. Thanks to the inherent parallelizability of the BF method, it is aptly suited for GPU implementations, leveraging both global and texture memory. However, a drop in performance was observed with global memory when memory accesses were non-coalesced, despite its high bandwidth. The study's findings underscore the efficacy of the CUDA API in accelerating the kNN search, achieving speeds up to 400 times faster than a comparable CPU-based BF approach. This highlights the substantial potential of GPUs in enhancing the efficiency of computing kNN in large d-dimensional datasets.

TBiS The use of GPUs, especially NVIDIA's, for multi-core parallel processing has gained traction in various projects due to their enhanced support for application interfaces [27, 81]. Many GPU implementations involve modifying or tailoring specific sorting algorithms to fit particular needs. Garcia et al., for instance, employed the insertion sort and a parallel comb sort in their work [27]. Bitonic sorting stands out as a preferred choice for parallel systems because its operations of reading, comparing, swapping, and writing during the sorting process are data-independent [82]. In a distinctive study by Sismanis et al. [83], they explore parallel techniques to find the k-nearest neighbors

(kNN) for individual queries in a high-dimensional space, specifically on a GPU. Their focus is on the brute force (BF) kNN sorting process. They introduced a set of truncated sort algorithms for parallel kNN searches, leveraging the close relationship between the select and sort operations. The truncated bitonic sort (TBiS) is highlighted for its straightforward data and program structures, efficient data locality, and synchronous concurrency. In TBiS, the overhead diminishes with each iteration, boasting a time complexity of $O(logn)$ for the parallel scan. The process begins with identifying the k-th element as a threshold, then examining all elements below this threshold. Subsequent searches are carried out to identify elements equal to the threshold, and any item confirmed not to be among the minimal k is excluded from the sort. The study underscores that both the Bubble Sort and Bitonic Sort methods feature data-independent synchronous processes, making them efficient for parallel processing tasks.

QDBI In the evolving landscape of peer-to-peer (P2P) systems that house a plethora of high-dimensional data including texts, images, and videos, the challenge of effectively searching through this extensive data has emerged as a pivotal area of research. The kNN query, a complex query type for high-dimensional data, has been scrutinized in numerous studies [84–86]. In this realm, a novel approach named distributed multi-dimensional data index (QDBI) has been proposed [87]. QDBI is anchored in the use of quad-trees, where each peer employs an MX-CIF quad-tree to create an index for their high-dimensional data. Each index item is then assigned a code following the MX-CIF quad-tree structure. The indices, characterized by their codes, are orchestrated into one-dimensional rings. Super-peers are dynamically integrated into these rings as per the demands, forming a semantically structured super-peer network. Within this structure, an efficient kNN query processing mechanism is engineered, specifically tailored for high-dimensional data items, leveraging the QDBI index framework. The research underscores the efficacy and scalability of both the kNN query approach and the index structure. When a super-peer receives a kNN query, it is initially directed to the super-peer that holds the corresponding control point (quad-tree block). Subsequently, the super-peers engage in a simultaneous evaluation of the query. Given that index items are arranged in the ring in a clockwise fashion according to the code value of their control points, the query might gather comprehensive solutions from one or multiple super-peers along the ring in the counterclockwise direction. The underlying principle is rooted in the gradual expansion of the search space as the query extends its range. In every expanded search space, identifying the k closest data points to the query point facilitates the updating of existing data across all spaces with the latest findings. The process culminates when the count of the closest data surpasses k and no closer data emerges in subsequent searches.

CU-kNN Reference [88] introduces the utilization of CUDA architecture for GPU parallelization in the realm of data mining applications. The authors unveil three innovative CUDA-based parallel algorithms aimed at optimizing data mining processes on parallel platforms. These comprise a scalable thread scheduling strategy tailored for irregular patterns, a parallel distributed top-k strategy for selecting top-k values efficiently, and a technique for parallel high-dimensional data reduction. Specifically focusing on the CU-kNN approach, the core of kNN computation lies in two pivotal elements: the selection of kNN and the calculation of distances. The CUDA implementation showcased in this

study ensures that all calculations are executed by the GPU, eliminating the need for data transfer between host and device memory during the classification phase. Given the independence of pair-wise distance computations, the authors capitalized on this trait to fully parallelize the process, making kNN ideally suited for GPU-based parallel execution. The kernel's design is focused on minimizing global memory accesses while amplifying the concurrency of distance computations executed across multiple threads. To manage the multidimensional nature of objects in the application, each object is partitioned into segments, and distances between these segments are computed iteratively. Threads collaboratively load feature segments of query and reference objects into shared memory, execute local calculations, and generate local summations. These summations are then aggregated to compute the final distance value. In the context of selecting the k nearest neighbors, the authors utilize the parallel distributed top-k strategy. Since the selection process for different query objects is independent, it is parallelized, with specific threads assigned to select the k shortest distances for individual query objects. Empirical assessments underscored the superiority of CU-kNN, highlighting its performance, which surpassed Fast-kNN [29] by a factor of up to 8.31 times on a real-world dataset from KDD-CUP 2004 quantum physics.

kNN-PA This study [89] introduces a specialized library for executing closest neighbor searches in high-dimensional datasets on thousands of cores, utilizing the Message Passing Interface (MPI) [90] and OpenMP [91]. The library is designed to deliver both exact and approximate results, focusing mainly on approximate nearest neighbors (ANNs) using tree indexing. The authors present two distributed brute-force kNN methods: two-dimensional partitioning and cyclic partitioning. The two-dimensional approach involves partitioning both reference and query points and allocating them to nodes. Although this technique consumes more memory due to replication of points, it proves efficient in scenarios where time is a critical performance factor. The cyclic partitioning technique, on the other hand, is less memory-intensive but demands more communication. It involves dividing reference and query sets into equal-sized segments, distributing them across processes, and cycling either set based on their size. Additionally, the study introduces a parallel tree-building method, RKDT (randomized k dimensional tree) [89], compatible with multiple tree types and capable of indexing structures in any dimensional space. The authors explore two point-splitting methods within this approach: cluster-based (RKDTC) and hyperplane-based (RKDTH). The former uses k-means clustering, while the latter involves projecting all points onto a computed projection direction. Hyperplane-based partitioning proves more efficient, offering equal data point division and optimal load balancing, whereas the cluster-based approach is more computationally demanding. Hyperplane splitting also enhances filtering during neighbor searches. For tree traversal, the authors contrast greedy traversal, where a query point explores one of the current node's child nodes, with bounding ball traversal, where every leaf node intersecting the bounding ball is visited. To address kNN challenges, a two-stage tree traversal method is employed. Initially, each query point uses greedy traversal to reach a leaf node and identify its kNN. Subsequently, a bounding ball search, with a radius equal to the kNN (k-th item distance), identifies all nearest neighbors within the bounding ball. The closest k items among the identified neighbors are then returned. This comprehensive approach ensures enhanced performance and

accuracy in high-dimensional datasets where bounding balls and leaf nodes extensively overlap.

HkNN In study [92], Muhr and Affenzeller developed a hybrid approach to enhance kNN Search performance on high-dimensional datasets. They identified that traditional indexing structures often fall short in such environments, leading to the necessity for an efficient brute-force search strategy. Their innovative method leverages both CPU and GPU, utilizing their distinct capabilities to optimize the search process. The hybrid approach is rooted in the synergy of the GPGPU paradigm [93] and CPU-based shared-memory parallelism to address the challenges posed by high-dimensionality in kNN Search. In this setup, the GPU is tasked with calculating distances, while the CPU, operating asynchronously, focuses on selecting the k nearest neighbors. Batching techniques enable these two processes to occur simultaneously, ensuring optimal efficiency. Remarkably, the effectiveness of the HkNN method scales linearly with the increase in dataset dimensions, offering a substantial performance advantage, especially for large datasets, over conventional methods.

kNN join approach

We have categorized the various kNN Join strategies into four distinct groups: I/O-based, main-memory-based, parallel, and distributed techniques.

IO-based

As an example, Böhm et al. [8] introduced a kNN Join issue that identifies the kNN for multiple queries in one operation. Research into kNN Join methods, including MuX (Multi-page Indexing) [8], Gorder (G-ordering kNN) [94], and iJoin [95], has incorporated the nested loop search approach for handling high-dimensional data sets.

MuX Böhm and Krebs [7, 8] introduced the concept of kNN Join and developed a new method known as multi-page indexing to execute it. This technique utilizes the index nested loop join approach and is based on the R-tree [15] structure. The researchers employed large-size pages (hosting pages) to reduce I/O time and introduced buckets, smaller minimum bounding rectangles, for more precise data division with reduced CPU expense. Each set of objects (R and S) has its own index. MuX scans the index pages on R, and for each R page stored in memory, it pulls the corresponding S pages using the index to find the kNN. The process concludes when all pages are either scanned or filtered. The performance of MuX is boosted by merging page-loading and bucket-selection strategies. However, it faces challenges such as performance decline with increased dimensionality and considerable memory overhead, limiting its scalability.

Gorder Gorder (G-ordering) [94] is a method that employs a block-nested loop join approach, optimizing both I/O and CPU costs through sorting, join scheduling, and distance computation. It consists of two main phases: PCA [30] and grid order sorting. PCA is used to identify the direction of greatest data variance. In grid order sorting, the data space is divided into rectangular cells, which are then ordered based on block distance, facilitating a scheduled block nested loop join on the G-ordered data. In Gorder, the G-ordered dataset is divided into blocks containing multiple physical pages. The method is characterized by its two-tier partitioning system that optimizes I/O and CPU times and its scheduling of data joining to improve kNN processing. The

first-tier involves loading R blocks sequentially and iteratively into the main memory, and for each loaded R block, an S block is loaded based on a scheduled sequence of similarity. However, this process can be CPU-intensive due to the large block size. To address this, the second-tier partition strategy is introduced, where blocks are divided into sub-blocks within the main memory, with each sub-block containing 20–25 points for efficient processing. Given the nature of high-dimensional data, minimizing distance calculation is crucial for optimizing CPU time. An algorithm is introduced to reduce distance computation. It starts by calculating the minimum distance between two blocks of G-ordered data, B_r and B_s. If this distance exceeds the pruning distance, the block is pruned, ensuring efficiency in processing. On the other hand, if the calculated minimum distance is smaller than the pruning distance, the block is not pruned. Instead, it is acknowledged as the k nearest neighbour, marking its significance in the search process.

iJoin Series iJoin [95] is a method developed to enhance performance in kNN computations, taking advantage of the iDistance index structure and partition strategy. This technique has been refined into three versions: the basic iJoin, iJoinAC, and iJoinDR, each offering unique improvements. In the base version, iJoin, datasets R and S are divided into clusters, each associated with the same reference points. Using two B^+-tree based iDistance indexes, the datasets are indexed efficiently. The method searches within a certain radius and prunes unnecessary partitions to find join pairs. If it does not find any, it expands the search area to include more S nodes. iJoinAC, an advanced version, aims to reduce the number of distance calculations and disk usage. This version is nearly identical to the original iJoin but focuses on processing approximation cubes instead of the actual feature vectors, making computations faster. The third variant, iJoinDR, incorporates dimensionality reduction to lessen I/O and CPU costs. It captures most of the information in the initial dimension using PCA and employs a sorting method for efficient approximation and filtering processes, similar to iJoin but more efficient thanks to these enhancements. However, all these methods [8, 94, 95] are applied to static datasets, meaning they require a complete kNN computation for all users during updates, leading to a significant processing cost.

IIB and IIIB Algorithms Reference [96] introduces three kNN Join algorithms specifically designed for handling high-dimensional and sparse datasets. The Brute Force (BF) algorithm is the foundational method, where each query point r in block B_r is compared with each object point s in block B_s based on their similarity scores. If the similarity score exceeds the existing pruning score of r, the object point s is identified as a nearest neighbor, and the pruning score is updated accordingly. To optimize this process, the Inverted Index-Based (IIB) algorithm is introduced. IIB is crafted to eliminate the need to traverse each item in the object dataset, addressing the inefficiencies of the BF algorithm. Instead of evaluating all features of the query point r, IIB prunes unnecessary features of s while calculating the dot product of r and s. This optimization is achieved using an inverted list $\{I_1, I_2, \ldots, I_D\}$, which contains sets of lists for each dimension, aiding in computing the kNN for each query point r in B_r more efficiently. Building upon the advancements of IIB, the Improved Inverted Index-Based (IIIB) algorithm incorporates a new threshold-based pruning method. In IIIB, previously calculated results are utilized as thresholds for subsequent iterations,

streamlining the kNN computation process even further for sparse vectors. These enhancements make the IIIB algorithm a more refined solution for tackling the kNN Join problem in high-dimensional, sparse datasets.

kNNJoin+ Yu, Cui, and colleagues [28] developed the kNNJoin+method to efficiently handle kNN Join queries in high-dimensional datasets, as outlined in their study. The process of executing an RkNN query is notably resource-intensive compared to a standard kNN query, so it is deployed selectively when the benefits outweigh the computational costs. In their innovative approach, the researchers employ four distinct data structures to dynamically update tables throughout all operations. These structures include the RkNN Join table, kNN Join table, iDistance, and sphere-tree. The sphere-tree plays a crucial role in identifying RkNN – points for which a particular point p is considered as one of the k nearest neighbors. On the other hand, iDistance indexing comes into play to determine the k nearest neighbors for any newly added point$p$. iDistance employs the Pyramid technique to convert the complex, multi-dimensional data space into a simplified, one-dimensional value, making the data more manageable. The technique partitions the data space, assigns a unique reference point to each segment, and then relates each section to a one-dimensional space according to the similarity of data points to the assigned reference point. These transformed values are then systematically organized into a B^+-tree for quicker access and efficient updates. To bolster performance further, the team crafted a shared query optimization strategy, ensuring that the process of handling kNN Join queries remains as efficient and streamlined as possible, even in the context of complex, high-dimensional data environments.

Memory-based

HDR-Tree Yang, Chong, and their team [34] introduced two data structures for handling high-dimensional data: the HDR-tree (for exact solutions) and the HDR*-tree (for approximations). The HDR-tree integrates the PCA [31] technique with clustering to reduce data dimensions. In contrast, the HDR*-tree opts for the Random Projection [97] strategy, which applies a random matrix to transform data dimensions. Essentially, PCA helps reduce the computational cost within the tree. The idea behind this is to focus on data directions with the highest variance, gradually picking the most significant components. The tree's initial dimension is packed with the most varied values, enabling more efficient data pruning and, subsequently, lower computational demands. Using eigenvalues, they tailor the dimensionality across tree levels. They partitioned the complex dataset into clusters and assigned varying dimensional depths to different tree levels. As you move deeper into the tree, dimensionality grows, reaching its full extent at the leaf nodes. The HDR-tree search algorithm zeroes in on specific users in a leaf node. For non-leaf nodes, it checks for a pruning condition and continues the search if criteria are met. If a certain distance condition is exceeded, certain clusters get bypassed, further optimizing the search process. This approach is particularly efficient for ongoing kNN Join operations on real-time, high-dimensional datasets, offering a way to lower in-memory search expenses.

EkNNJ In real-world scenarios, databases are frequently updated with the addition or removal of items. However, a gap exists in the capability of current kNN Join algorithms to efficiently handle batch updates and deletions, according to a study [98]. To

fill this gap, the authors introduced EkNNJ, an enhanced method to perform kNN Joins on high-dimensional data, with a focus on batch processing, delayed updates, and optimized deletion operations. In EkNNJ, when an item is added or removed, the affected users are identified but not immediately updated. They are marked as "dirty" nodes in the HDR-tree and only updated when necessary, a strategy known as lazy updates. This is particularly useful as many new items often affect the same users, leading to redundant calculations if updated each time. Furthermore, EkNNJ employs batch operations to enhance efficiency. Instead of updating the affected users with every new item insertion or deletion, the updates are accumulated and processed together, reducing computational efforts. Deletion operations in kNN can be complex and resource-intensive. Traditionally, every affected user needs to be identified and their kNN list updated whenever an item is deleted. EkNNJ simplifies this process by maintaining a reversed kNN (RkNN) table for all items, which speeds up the identification of affected users and their subsequent updates. In summary, EkNNJ stands out for its efficiency in handling dynamic update operations, especially batch updates and deletions, offering a more resource-efficient alternative to existing methods like HDR-tree and naive RkNN.

Parallel and distributed

kNN Join operations can be quite resource-intensive when dealing with high-dimensional data. While various efficient strategies have been devised, they are typically designed for a single machine or thread. However, parallel and distributed computing methods have gained traction for their ability to enhance performance. MapReduce [99] is particularly notable in this context, acclaimed for its simplicity and effectiveness in parallel and distributed processing, offering a solution to the limitations of main memory when handling large datasets. Specific parallel MapReduce kNN Join algorithms such as Ball Tree [100], k-d Tree [19], PAT[101], HBNLJ [102], H-BRJ [102], and PGBJ [12] have been developed, underscoring the framework's utility in efficiently processing substantial amounts of data in parallel.

Ball Tree This algorithm [100] utilizes a tree structure, specifically a ball tree, for efficient nearest-neighbor searches. The effectiveness of the ball tree depends on the data distribution and query type. It employs a distance metric, like Euclidean distance, to expedite the identification of the k nearest points to a specified test point. A notable feature of this algorithm is its ability to bypass any subtree whose distance from the test point exceeds that of the closest identified point, enhancing efficiency. The algorithm initiates from the root node, proceeds depth-first, and keeps a record of the k nearest points in a max-first priority queue. Three possible operations can occur at each node. If the distance to the test point surpasses that of the furthest point in the queue, the node is skipped. For leaf nodes, each point within is evaluated, and the queue is updated accordingly. Internal nodes invoke a recursive search on their descendants, prioritizing the one closer to the test point, thereby optimizing the queue's updates and often eliminating more distant nodes from consideration.

k-d Tree The k-d Construction Algorithm [19] facilitates the creation of a k-d tree, enabling efficient nearest neighbor identification. This offline, top-down recursive algorithm divides data points into two groups based on the median value along the dimension with the largest spread. The Nearest Neighbor (NN) algorithm, leveraging the k-d

tree's structure, swiftly narrows down the search space. It descends recursively from the root, comparing the search point to nodes along the dividing dimension, until reaching a leaf node where the distances are compared. The current best distance is updated if a shorter one is found. As the algorithm ascends back, it checks if points on the other side of the dividing plane might be closer. If potential closer points are found, the same process recurs on the opposite branch of the tree. Upon reaching the root again, the closest point found is declared the nearest neighbor. The construction of the k-d tree has a time complexity of $O(nlogn)$, where n represents the total number of data points.

PAT The Principle Axis Tree Nearest Neighbor (PAT) algorithm [101] enhances search efficiency by constructing a search tree through principal component analysis (PCA). It segments the dataset into regions, focusing on the node with maximal variance. Utilizing a depth-first search and a unique elimination criterion, it effectively pinpoints the k nearest neighbors. The dataset is projected onto the principal axis, recursively divided into equal subsets until each contains a minimal number of data points. The depth-first search starts at the root node, using binary search to identify the query point's region, repeating this until the final node is reached. Here, partial distance search (PDS) measures the distances between points within that node. The elimination criterion is applied to the closest sibling node, potentially leading to its examination or removal. The process either investigates or eliminates sibling nodes before reverting to the root, optimizing the search operation.

H-BNLJ and H-BRJ In this study, two methods for handling large datasets were introduced, both utilizing the MapReduce framework [102]. The first method, H-BNLJ (Hadoop Block Nested Loop Join), involves dividing datasets R and S into equal-sized blocks during the mapping phase. These blocks are then paired, divided into buckets, and processed by reducers to find the k nearest neighbors (kNN) using a nested loop. However, this basic approach faced scalability issues with high-dimensional data. To enhance this, a refined method, H-BRJ (Hadoop Block R-tree Join), was introduced. In this improved version, an R-tree based index is constructed for each local S block within a bucket. This index aids in the efficient search for kNN within the same bucket, eliminating the need for a local nested loop and thereby saving computational resources. The R-tree enables quicker kNN searches, making the H-BRJ a more efficient alternative for processing large, multidimensional datasets. Each reducer's output, containing records' ids and the distances to their respective kNNs, is stored in a distributed file system (DFS), sorted, and the top-k results for each record id are released. This method offers a more scalable solution for handling complex, large-volume datasets in a parallel computing environment.

PGBJ Reference [12] is a method that utilizes Voronoi diagrams to divide the entire data space into cells, assigning data to the closest pivot within each cell. However, it sometimes requires searching through multiple cells to find the k nearest neighbors (kNN), leading to data duplication and increased computations. The author proposes two strategies to group cells into larger cells to address this issue: Geo grouping (grouping adjacent cells) and Greedy grouping (grouping cells with a high duplication probability). Despite its low communication overhead and disk usage, PGBJ's efficiency diminishes with high-dimensional datasets. The process of selecting pivots, crucial for PGBJ's performance, can also be time-consuming, especially with larger datasets.

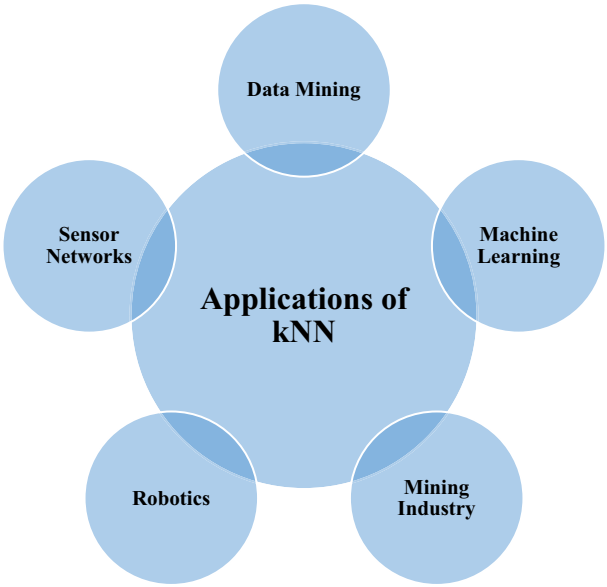


Fig. 5 Applications of kNN

Applications of kNN

The k-Nearest Neighbors (kNN) algorithm is widely used across various domains for its simplicity and effectiveness. Figure 5 shows the applications of kNN.

Data mining [69, 103–112]

kNN is extensively used in data mining for classification and regression tasks. For instance, in [69], the authors utilized kNN for credit scoring, demonstrating its effectiveness in distinguishing between good and bad credit risks based on historical financial data. Another notable application is seen in [103], where kNN was employed to predict customer churn by identifying patterns in customer behavior data. Additionally, [104–108] discussed the use of kNN in fraud detection, where it helped in identifying fraudulent transactions by analyzing transaction patterns. In [109, 110], kNN was applied to medical diagnosis, specifically for classifying patient data into different disease categories based on symptom similarity. Lastly, [111, 112] highlighted its use in market segmentation, where kNN helped in grouping consumers with similar purchasing behaviors.

In machine learning, kNN is used as a classification or regression algorithm where the focus is on predicting the output for a new data point based on its nearest neighbors. In contrast, kNN search in data mining refers to the process of finding the nearest neighbors in a dataset, often used for tasks like clustering or similarity search, without necessarily performing classification or regression.

Machine learning [113–125]

In the field of machine learning, kNN serves as a fundamental algorithm for various supervised learning tasks. For example, [113, 114] explored its use in image recognition,

where kNN classified images based on pixel intensity similarities. In [115, 116], kNN was used for text categorization, classifying documents into topics based on word frequency features. Moreover, [117–121] highlighted its application in recommendation systems, where kNN recommended products to users based on the similarity of user profiles. Another study, [122, 123], illustrated how kNN can be used for anomaly detection in network traffic, helping identify unusual patterns that may indicate security breaches. In [124, 125], kNN was employed in bioinformatics for gene expression analysis, classifying genes based on expression levels to understand disease mechanisms.

In machine learning, kNN is used to classify or regress data points by considering the labels of their nearest neighbors. On the other hand, kNN search in the survey context is about efficiently finding the nearest neighbors in high-dimensional spaces, which is a fundamental operation that can support various tasks like similarity search or data retrieval, rather than direct classification or regression.

Machine industry [126–130]

The machine industry benefits from kNN in areas such as predictive maintenance and quality control. In [126], kNN was used to predict equipment failures by analyzing sensor data from machinery, allowing for timely maintenance and reduced downtime. Another application in [127] focused on quality inspection, where kNN helped classify products based on visual inspection data to ensure they meet quality standards. Additionally, [128–130] discussed the use of kNN in optimizing manufacturing processes by analyzing historical production data to improve efficiency and reduce waste.

In machine learning, kNN in the machine industry typically refers to using the algorithm to predict outcomes like failures or classify product quality based on nearest neighbor analysis. kNN search, however, is about identifying similar instances within large datasets, which supports these predictive or classification tasks by quickly finding relevant data points without necessarily performing the prediction or classification itself.

Robotics [131–136]

In robotics, kNN is applied for object recognition, path planning, and control systems. For example, [131–133] used kNN to enable robots to recognize and classify objects in their environment based on visual input. In [133], the authors applied kNN for autonomous navigation, where the algorithm helped the robot select the optimal path by comparing current sensor data with stored navigation maps. Additionally, [134] explored kNN for robotic arm manipulation, assisting in precise movement control by referencing past movement data to achieve desired positions. Another significant application is seen in [135, 136], where kNN was used for human-robot interaction, helping robots understand and respond to human gestures and commands.

In robotics, kNN in machine learning is primarily used for making decisions based on the classification or regression of sensor data, such as object recognition or path planning. The kNN search, however, focuses on finding the nearest data points to a query point, which can be used to support these tasks by providing relevant data for decision-making processes.

Sensors network [137–142]

kNN is pivotal in sensor networks for tasks like environmental monitoring and localization. In [137], kNN was employed to analyze temperature and humidity data from a network of sensors to predict weather patterns. Another significant application in [139] involved using kNN for indoor localization, where it estimated the position of a device by comparing real-time signal strength data with a pre-recorded signal map. Additionally, [140] used kNN for detecting anomalies in sensor data, which is crucial for maintaining the reliability of the sensor network by identifying faulty sensors or unusual environmental conditions. In [141, 142], kNN was applied to smart home systems, where it helped in monitoring and controlling home environments based on sensor inputs.

In sensor networks, kNN in machine learning is typically used for classification or regression tasks based on sensor data, such as predicting weather patterns or detecting anomalies. kNN search, on the other hand, involves finding the nearest neighbors in a dataset, which can be used to support these applications by quickly identifying relevant data points for further analysis. kNN's multifaceted applications underscore its versatility and efficacy in diverse domains. Its adaptability, ease of implementation, and precision make it a favored choice for professionals and researchers alike, driving innovations and enhancements in data mining, machine learning, the mining industry, robotics, and sensor networks. The algorithm's capacity to handle complex, high-dimensional data and deliver precise, reliable outcomes underscores its integral role in advancing technology and data-driven solutions.

Comparative analysis

In this segment, a detailed comparison is presented between cutting-edge kNN Search and kNN Join methodologies, specifically applied to high-dimensional datasets. This analytical comparison aids in achieving a deeper insight into each approach, illuminating the distinct features and potential shortcomings inherent in each.

kNN search approach

The diverse kNN Search strategies are thoroughly examined and compared in Table 4. This organized presentation of information facilitates an enhanced comprehension of the capabilities and constraints associated with each method, guiding informed decision-making and potential enhancements in future applications.

1. Classical kNN is appreciated for its simplicity and versatility in handling different data distributions and types of problems including classification and regression. It is particularly beneficial for datasets with multiple features and dimensions. However, its effectiveness is hampered by the high computational cost associated with large datasets and increased dimensions. The choice of k is crucial, as an inappropriate value can lead to overfitting or underfitting. While it is a robust option for smaller datasets and can handle multiple features efficiently, it can suffer in performance due to the curse of dimensionality in larger and more complex datasets. (Source Code: https://github.com/rajib1346/kNN/blob/main/Classical_kNN.py)

Table 4 Pros. and Cons of kNN Search

Techniques	Pros.	Cons.	Datasets and performances
Classical kNN [69]	1. The algorithm is straightforward and makes no assumptions about the data distribution 2. It can be used to solve problems involving classification and regression 3. It can be advantageous for datasets with numerous characteristics and high dimensionality	1. Large datasets and high-dimensional feature spaces impose a high computational cost 2. Sensitivity to the selection of k, resulting in overfitting or underfitting 3. As the number of features or dimensions increases, efficacy degrades due to the curse of dimensionality	Dataset: Iris Accuracy: 100% Training Time: 0.00 s Testing Time: 0.05 s
W-kNN [70]	1. The accuracy of the method is enhanced by assigning distance-based weights to the k Nearest Neighbors 2. The method is resistant to the influence of data outliers and noise 3. There are numerous weight functions and distance metrics from which to choose	1. WkNN is computationally expensive, making it unsuitable for large datasets 2. Selecting a suitable weighting function for WkNN can be challenging 3. WkNN is sensitive to outliers in the dataset, which can lead to incorrect predictions	Dataset: Iris Accuracy: 100% Training Time: 0.0009 s Testing Time: 0.07 s
CNN [71]	1. CNN can substantially reduce the original dataset's size while maintaining classification accuracy 2. CNN can enhance the performance of classification algorithms by removing instances that are chaotic and irrelevant from the dataset 3. By utilizing a reduced subset of examples for training, CNN can reduce the training time of classification algorithms. The rate of recognition	1. If the selected subset is too small or not representative of the original dataset, CNN can result in overfitting 2. CNN is sensitive to the initial subset selected and the order in which instances are evaluated 3. CNN may not be appropriate for datasets with intricate decision boundaries or classes that overlap	Dataset: Iris Accuracy: 90% Training Time: 0.001 s Testing Time: 0.16 s
RNN [72]	1. Because it reduces the computational complexity of the nearest neighbor algorithm, the RNN algorithm is an effective method for processing large datasets 2. RNN can enhance the scalability and efficacy of the nearest neighbor algorithm while preserving classification accuracy 3. RNN can be utilized with any distance metric, not just the Euclidean distance	1. If insufficient proximal neighbors are evaluated, RNN may reduce the classification accuracy 2. The efficacy of the RNN algorithm can be considerably impacted by the number of nearest neighbors that are considered, indicating that choosing the appropriate number is crucial 3. RNN may not be appropriate for datasets with intricate decision boundaries or classes that overlap	Dataset: Iris Eigen Vector Transformation: Yes Compressed: Yes k = 3 Error Probability: 0.0 Eigen Vector Transformation: No Compressed: No k = 3 Error Probability: 0.88
ENN [73]	By eradicating instances that are incorrectly classified or contain noise, the ENN algorithm can substantially improve the classification accuracy of the nearest neighbor algorithm 2. ENN can be used as a preprocessing phase to enhance the efficiency of other classification algorithms 3. ENN is a straightforward and efficient algorithm that can be readily implemented and applied to diverse datasets	1. ENN may eliminate some pertinent instances from the dataset, resulting in a loss of information 2. The number of nearest neighbors considered and the order in which instances are edited can have a significant impact on the efficiency of ENN 3. ENN may not be appropriate for datasets with intricate decision boundaries or classes that overlap	Dataset: Pima Diabetes Mean Accuracy: 0.7535 Mean Precision: 0.7346 Mean Recall: 0.7122

Table 4 (continued)

Techniques	Pros.	Cons.	Datasets and performances
SNN [74]	1. Unlike many other clustering algorithms, the SNN algorithm is able to process high-dimensional data 2. It can recognize non-linear structures in the data and can deal with noise 3. Compared to other clustering algorithms, its performance on a variety of datasets has been demonstrated to be superior	1. The outcome of clustering is affected by the selection of parameters, such as the threshold value and the number of nearest neighbors; therefore, it may be necessary to modify the parameters for each dataset to achieve optimal results 2. The algorithm may be computationally costly, particularly for large datasets 3. The algorithm is sensitive to the distance metric chosen and may not function well with certain distance metrics	Dataset: Unknown Query: Branching carbon more than 13 Accuracy: 86%
VBNN [75]	1. The VBNN algorithm is easy to implement and understand, making it accessible to beginners in machine learning 2. The algorithm is effective for datasets with well-separated clusters, as it can accurately classify new data points based on their proximity to Voronoi boundaries 3. The VBNN algorithm can be used for both clustering and classification tasks, making it versatile in its applications	1. The VBNN algorithm is unsuitable for datasets with clusters that overlap, as it may misclassify data points that lie within multiple Voronoi regions 2. The algorithm is computationally intensive for large datasets due to the construction and modification of the Voronoi diagram for each new data point 3. The choice of the distance metric used to calculate the distance between data elements has an effect on the accuracy of the VBNN algorithm	Datasets: NavTech, USGS Entries: Hospitals Query processing time: 6.5 s Entries: Shopping centers Query processing time: 3.3 s Entries: Parks Query processing time: 1.5 s Entries: Schools Query processing time:0.75 s Entries: Auto services Query processing time: 0.65 s Entries: Restaurants Query processing time: 0.57
M-kNN [76]	1. The algorithm decreases the necessary quantity of data points to achieve classification, which results in higher efficiency 2. It eliminates reliance on the k-value, making it more adaptable to various datasets 3. Using representative data points reduces the influence of chaotic data on classification outcomes	1. The algorithm may not perform optimally with datasets containing class boundaries that are intricate and overlap 2. The procedure of selecting representative data points can be time-consuming and expensive computationally 3. Choosing the appropriate representative data elements can significantly affect the quality of the classification results	Dataset: Glass Accuracy:6857% Dataset: Iris Accuracy: 95% Dataset: Heart Accuracy: 89.74% Dataset: Wine Accuracy: 95.43% Dataset: Diabetes Accuracy: 74.77% Dataset: Aust Accuracy: 86.9%

Table 4 (continued)

Techniques	Pros.	Cons.	Datasets and performances
R-kNN [77]	1. Incorporates a ranking model to promote the trustworthiness of neighbors' labels, potentially improving the accuracy of predictions 2. Can be incorporated with any distance metric to provide some degree of improvement 3. Provides a performance metric using Hamming loss to evaluate the algorithm's performance	1. Requires the training of a ranking model, which may be computationally expensive 2. The optimization method used to learn the weights for the weighted voting strategy may not always result in the optimal solution 3. The algorithm may not perform well in high-dimensional spaces or with noisy data	Hamming Loss results: Dataset: yeast RK + GPS: 0.19 Dataset: scene RK + GPS: 0.08 Dataset: emotions RK + LW: 0.1860 Dataset: audio RK + GPS: 0.085 Dataset: genbase RK + GPS: 0.0011 Dataset: medical RK + GPS: 0.0117
C-kNN [78]	1. By clustering the training samples, the problem's complexity is reduced and the algorithm's efficacy is enhanced 2. The introduction of weight values enhances the accuracy of the algorithm by giving greater weight to clusters with more samples 3. The C-kNN algorithm is capable of managing complex and multi-peaked data distributions, making it suitable for a variety of classification tasks	1. The clustering process may not always produce an optimal outcome, and the quality of the clusters is dependent on the clustering algorithm and parameters selected 2. The C-kNN algorithm may be computationally expensive, especially when working with large datasets or high-dimensional feature spaces 3. The algorithm's efficacy may depend on the characteristics of the dataset, and it may not consistently outperform other classification algorithms in all situations	Dataset: text classification dataset Avg. Precession: 0.9133 Avg. Recall: 0.9115 Avg. F1: 0.9123
NFL [79]	1. The NFL algorithm can handle variations within a class by capturing the subspace of feature variations, making it effective for image classification and retrieval 2. It can potentially improve the accuracy and efficiency of traditional nearest neighbor search methods by utilizing the subspace information 3. The algorithm does not require any training or additional data, making it a simple and straightforward approach for image retrieval	1. Assuming that each class has a well-defined subspace, the NFL algorithm may not perform well on datasets containing complex or overlapping classes 2. Construction of the subspace can be computationally intensive, especially for large datasets with numerous prototypes and features 3. The efficacy of the NFL algorithm is highly dependent on the quality of the selected prototypes, which can be difficult to ascertain in some datasets	Dataset: Brodatz texture Error rate: 5.3 Dataset: Color image Error rate: 44.4 Dataset: Image using Gabor Error rate: 7.9 Dataset: Wavelet features Error rate: 42.0
LNN [80]	1. It can handle non-linear decision boundaries 2. It can model complex relationships between features and labels 3. This method is appropriate for use with datasets containing a large number of dimensions	1. The effectiveness of the method can be influenced by the choice of hyperparameters, including but not limited to the distance metric used and the number of neighbors considered 2. It can be computationally costly for massive datasets 3. If the training dataset is too limited or noisy, it is susceptible to overfitting	Dataset: Face image Recognition rate: 95.18 Recognition time: 0.027 s

Table 4 (continued)

Techniques	Pros.	Cons.	Datasets and performances
BF-CUDA [29]	Enables quick, concurrent kNN Search	Ineffective with extremely large datasets	Dataset: Unknown d = 96, n = 1200 Computation time: 0.02 s n = 2400 Computation time: 0.05 s n = 4800 Computation time: 0.15 s n = 9600 Computation time: 0.57 s n = 19,200 Computation time: 2.29 s n = 38,400 Computation time: 9.61 s
TBiS [83]	Features straightforward data/programme configurations, simultaneous operations, superior data localization	Efficiency plummets as data volume and k value expand	Dataset: BelgaLogos Computation time: 1.3 to to 4 s
QDBI [87]	Exhibits excellent scalability and search efficacy	Testing omitted real datasets, limited applicability to high-dimensional data	Dataset: Postal address dataset As the K value rises, both algorithms show a gradual increase in visited peers. The KNN query processing in SSW has a higher number of visited peers compared to the SCBO network
CU-kNN [88]	Delivers top-tier performance, enhances fundamental issues in CUDA-based data mining algorithms	Struggles with handling large data volumes, data transfer expense is problematic	Dataset: KDD-CUP 2004 Execution time: 8.31 s
kNN-PA [89]	Scalable, supports HD datasets kNN Searches utilizing thousands of cores	Not tailored for ongoing point modifications, falls short in distributed memory settings	Dataset: Synthetic dataset k = 8 Accuracy:99.50%±2.16%
HkNN [92]	Efficient CPU and GPU task distribution, hybrid methodology, performance boost with dimensionality increase	Evaluation only included brute-force techniques, opportunity for optimizing k selection and distance kernels	Dataset: High dimensional dataset
iDistance [42, 43]	Enables real-time query responses, adaptable to diverse data distributions, integrates into DBMS	Extensive search area, false negatives from lossy transformations, efficiency in pruning diminishes with rising dimensions	Dataset: High dimensional dataset Dimension: 30 k = 20 Accuracy: 100%
Diagonal Ordering [44]	Evades superfluous distance calculations, adaptable to assorted data distributions, supports real-time queries	Testing constrained to 30D dataset, concerns about efficacy in higher dimensions	Dataset: Image dataset CPU Cost: 200–300
VA^{+}-file [47]	Averts extreme data distribution disparities, suitable for non-uniform datasets	Requires approximation for results, efficiency dwindles with increasing dimensions, overlooks cache and query workload	Dataset: Airphoto The speedup for D = 1.05 is 4 The speedup for D = 1.1 is 6.9

Table 4 (continued)

Techniques	Pros.	Cons.	Datasets and performances
OTI and EOTI [48]	Quick search algorithm with reduced complexity, mitigates extensive distance calculations	OTI has extensive space and time requirements, triangle construction process could be improved	Datasets: Arcene, Multiplefeatures, GasSensors, Spambase, Waveform, DUWWTP, Shuttle, Slice Avg. PRST (KMC-OTI-FS): 52.28% Avg. PRST (KMC-EOTI-FS): 66.78%
BP [52]	Superior kNN Search, amplifies CPU time and IO cost efficiency	Lacks support for substantial data updates, conversion to L2 could yield improved solutions	Datasets: Real (Audio, Fonts, Deep, Sift), synthetic (Normal, uniform) datasets Running time: 525 ms I/O cost: 8.2×10^2
Δ-tree [26]	Enhanced index structure, condenses search space and accelerates queries in main memory	Tailored for correlated datasets, requires frequent complete tree reconstructions	Dataset: Corel image Elapsed time: 0.06 s
array-index [53]	Minimal disk access, fast and compact	Excludes real-world datasets in testing	Dataset: Image dataset Search time: 100 s
Δ^+-tree [26, 54]	Reduces computational expenses and cache misses, superior to iDistance and Pyramid tree	Incapable of halting tree reconstruction, parameter values fluctuate with dataset variations	Dataset: Motion capture, color histograms dataset Elapsed time: 2000s
ACDB [55]	Uses triangle inequality for reduced CPU expenses and enhanced performance	Initial center pivots and k value profoundly influence the clustering approach	Dataset: Bioretina Response time: 80 s
iDistance-PS [56]	Bolsters filtering capacity, efficient kNN query execution	Intensified dimensionality issues in space-oriented methods	Dataset: Uniform dataset
PL-Tree [62]	Adapts to dimensionality and data volume, supports point and range queries	Assessed with a 12D dataset only, iDistance surpasses in point query performance	Datasets: Synthetic, real world (USPP, TIGER, LLMPP, MAPS) dataset Query time (Synthetic dataset): 0.62 s Query time (LLMPP): 0 s Query time (TIGER): 3.6 s
iDStar [63]	Excels in clustered, high-dimensional data spaces	Reduced efficiency for dispersed clusters, inadequate for dimensions exceeding 256	NA
HC–O [66]	Enhances refinement speed during searches, applicable to both exact and approximate strategies	Assumes consistent query distribution, distance bound tightness and histogram impact pruning capability	Dataset: SOGOU Avg. query response time: 0.8 s

2. W-kNN enhances the accuracy of predictions by applying weights based on distances to the k Nearest Neighbors, making it more resilient to outliers and noise. There is a flexibility in choosing weight functions and distance metrics. However, its effectiveness can be challenged by the computational intensity, especially with large datasets. Choosing an optimal weighting function is also non-trivial and requires careful consideration. Although W-kNN can offer improved accuracy, especially in noisy environments, its computational expense and sensitivity to outliers can sometimes limit its applicability. (Source Code: https://github.com/rajib1346/kNN/blob/main/W-kNN.py).
3. CNN stands out for its ability to reduce dataset size while preserving classification accuracy, enhancing the performance of classification algorithms by filtering out irrelevant data. This makes it efficient in terms of training time and computational resources. However, its effectiveness can be compromised if the selected subset is too small or unrepresentative, leading to overfitting. It is also sensitive to the initial subset selection and the sequence in which instances are processed. While CNN can be a powerful tool for improving classification performance, care must be taken to ensure that the selected subsets are representative and adequately sized. (Source Code: https://github.com/rajib1346/kNN/blob/main/CNN.py)
4. RNN is noted for its capability to handle large datasets efficiently by reducing the computational complexity of the nearest neighbor algorithm. It maintains classification accuracy and offers flexibility in choosing distance metrics. However, its effectiveness is contingent upon the number of proximal neighbors considered—too few can lead to reduced accuracy. Similar to CNN, RNN may also struggle with datasets having complex decision boundaries or overlapping classes. RNN offers a scalable solution for large datasets but requires a judicious selection of neighbors to maintain accuracy. (Source Code: https://github.com/rajib1346/kNN/blob/main/RNN.py).
5. ENN xcels in enhancing classification accuracy by eliminating incorrectly classified or noisy instances, serving as a useful preprocessing step for other classification algorithms. It is straightforward and applicable to a variety of datasets. However, there is a risk of losing valuable information if pertinent instances are removed. The algorithm's efficiency is influenced by the number of neighbors considered and the editing sequence. It may also struggle with datasets having complex decision boundaries or overlapping classes. ENN can be a potent tool for noise reduction and improving classification accuracy but must be applied with caution to avoid information loss and to ensure it is suited to the dataset's complexity. (Source Code: https://github.com/rajib1346/kNN/blob/main/ENN.py).
6. SNN is notable for its capacity to process high-dimensional data, recognize nonlinear structures, and manage noise effectively. It has showcased superior performance across diverse datasets compared to other clustering algorithms. However, its effectiveness can be hampered by the need to fine-tune parameters like threshold value and number of nearest neighbors, and it may incur high computational costs with large datasets. The choice of distance metric is critical, requiring careful consideration to ensure optimal clustering results. (Source Code: https://github.com/rajib1346/kNN/blob/main/SNN.py)

7. VBNN is valued for its simplicity, ease of implementation, and effectiveness in datasets with well-separated clusters. It is versatile, catering to both clustering and classification tasks. However, its effectiveness is diminished with overlapping clusters due to the inherent limitations in handling data points lying within multiple Voronoi regions. Computational intensity and the influence of the chosen distance metric on accuracy are also areas of concern, suggesting that VBNN is best suited for specific dataset configurations and may not be a universal solution. (Source Code: https://github.com/rajib1346/kNN/blob/main/VBNN.py).
8. M-kNN stands out for its efficiency in classification by reducing the number of data points needed and eliminating reliance on k-value, making it adaptable. It counters the influence of chaotic data by using representative data points. However, its performance can be suboptimal with complex, overlapping class boundaries. The process of selecting representative data points is both time-intensive and computationally expensive. The algorithm's effectiveness, therefore, hinges on the careful selection of these points and may be more suited for datasets with distinct, well-defined classes. (Source Code: https://github.com/rajib1346/kNN/blob/main/M-kNN.py).
9. R-kNN enhances prediction accuracy by incorporating a ranking model and can be paired with any distance metric. Its performance is evaluable using Hamming loss. However, the need to train a ranking model adds to computational expense. The optimization method's effectiveness and the algorithm's performance in high-dimensional or noisy data scenarios can be uncertain. R-kNN promises enhanced accuracy in specific scenarios but requires adequate computational resources and careful parameter tuning to realize its potential. (Source Code: https://github.com/rajib1346/kNN/blob/main/R-kNN.py)
10. C-kNN optimizes efficiency by clustering training samples, with weight values enhancing accuracy. It is adept at handling complex and multi-peaked data distributions. However, the clustering outcome's optimality and computational expense, especially with large or high-dimensional datasets, can be potential drawbacks. The algorithm's effectiveness is contingent on dataset characteristics, suggesting its application should be tailored to align with dataset specificities for optimal performance. (Source Code: https://github.com/rajib1346/kNN/blob/main/C-kNN.py).
11. NFL is adept at handling class variations and is particularly suited for image classification and retrieval. It offers simplicity and does not require training or additional data. However, its effectiveness can be compromised in the presence of complex or overlapping classes, and the subspace construction can be computationally intensive. The selection of prototypes is critical, indicating that NFL's applicability is optimal in scenarios where class variations are well-defined and manageable. (Source Code: https://github.com/rajib1346/kNN/blob/main/NFL.py).
12. LNN is prized for its capability to handle non-linear decision boundaries and model complex feature-label relationships, even in high-dimensional datasets. However, its effectiveness is influenced by hyperparameter choices, computational costs with large datasets, and susceptibility to overfitting with limited or noisy training data. LNN offers a robust solution for complex, high-dimensional datasets but necessitates careful hyperparameter tuning and adequate training data to mitigate overfit-

ting and optimize performance. (Source Code: https://github.com/rajib1346/kNN/blob/main/LNN.py)

13. BF-CUDA is praised for its ability to quickly perform kNN Searches, thanks to its parallel processing capabilities. However, its effectiveness is limited by its inability to handle extremely large datasets. This algorithm is highly efficient for medium-sized datasets but might not be the go-to solution for large-scale applications. (Source Code: https://github.com/rajib1346/kNN/blob/main/BF-CUDA.py)
14. TBiS offers advantages like simplicity in data and program structures and optimal data localization. However, its performance diminishes with the increase in data items and k value. It could be an excellent choice for applications with moderate data volume and k values. (Source Code: https://github.com/rajib1346/kNN/blob/main/TBiS.py).
15. QDBI is recognized for its scalability and search performance but falls short as it has not been tested on real datasets and lacks focus on high-dimensional data. This makes assessing its effectiveness in real-world, high-dimensional applications challenging. (Source Code: https://github.com/rajib1346/kNN/blob/main/QDBI.py).
16. CU-kNN delivers high performance and addresses core issues associated with CUDA-based data mining algorithms but faces challenges with big data scalability and data movement costs. It might be suitable for applications where dataset size is not extensively large. (Source Code: https://github.com/rajib1346/kNN/blob/main/CU-KNN.py).
17. kNN-PA excels in scalability and is capable of handling high-dimensional datasets over thousands of cores but is not designed for continuous point updates and is less effective in distributed memory environments. It is particularly effective for static, high-dimensional datasets. (Source Code: https://github.com/rajib1346/kNN/blob/main/kNN-PA).
18. HkNN boasts efficient task distribution between CPU and GPU and offers a hybrid approach to kNN Search. However, its evaluation parameters are limited, and there is room for exploration in optimized k selection and distance kernels. Its effectiveness increases linearly with dimensionality. (Source Code: https://github.com/davnn/ParallelNeighbors.jl).
19. iDistance supports real-time queries and adapts well to different data distributions but suffers from a wide search region and lossy transformation issues. Its adaptability makes it versatile, though there are concerns about accuracy and efficiency in higher dimensions. (Source Code: https://github.com/rajib1346/kNN/blob/main/iDistance.py).
20. Diagonal Ordering avoids unnecessary distance computation and adapts well to varied data distributions but is limited by the scope of tested datasets. Its effectiveness in very high-dimensional datasets remains uncertain. (Source Code: https://github.com/rajib1346/kNN/blob/main/Diagonal%20Ordering.py).
21. VA+-file mitigates data distribution disparities and is useful for non-uniform datasets but faces challenges in approximation and performance degradation with dimensionality increase. It could be beneficial for specific non-uniform datasets but might struggle with high-dimensional data. (Source Code: https://github.com/rajib1346/kNN/blob/main/VA%2B-File.py)

22. OTI and EOTI algorithms offer reduced complexity and faster search but face challenges in space and time complexity, and efficiency in triangle construction. Their effectiveness may be contingent upon the dataset's nature and size. (Source Code: https://github.com/rajib1346/kNN/blob/main/OTI%20EOTI).
23. BP introduces a new approach to kNN Search and improves CPU and IO cost efficiency but struggles with massive data updates. Its effectiveness in real-time, dynamic environments may be compromised. (Source Code: https://github.com/rajib1346/kNN/blob/main/BP.py).
24. Δ-tree offers an optimized index structure and reduces search space but is only effective for correlated datasets and requires frequent rebuilding. It might be highly effective for specific dataset types but is not a universal solution. (Source Code: https://github.com/rajib1346/kNN/blob/main/%E2%88%86-tree.py).
25. array-index with minimal disk access and enhanced speed, array-index seems promising but has not been tested on real-life datasets. Its real-world effectiveness remains speculative. (Source Code: https://github.com/rajib1346/kNN/blob/main/array-index.py)
26. Δ+-tree minimizes computational cost but is hampered by the inability to halt tree-rebuilding processes and varying optimal parameter values. Its effectiveness may be high but requires specific tuning for each dataset. (Source Code: https://github.com/rajib1346/kNN/blob/main/%E2%88%86%2B-tree.py).
27. ACDB effectively reduces CPU cost but is significantly impacted by the initial center pivots and k values. It can be highly effective with the right parameter setting but requires careful tuning. (Source Code: https://github.com/rajib1346/kNN/blob/main/ABCD.py)
28. iDistance-PS enhances the filtering power of iDistance but faces significant issues with dimensionality in space-based approaches. Its effectiveness is enhanced in filtering but could be constrained by the dataset's dimensionality. (Source Code: http://code.google.com/p/idistance/).
29. PL-Tree scales well with dimensionality and data size but has only been tested on a 12D dataset. Its broad applicability and effectiveness are yet to be fully ascertained. (Source Code: https://github.com/rajib1346/kNN/blob/main/PL-Tree.py).
30. iDStar performs excellently in high-dimensional, tightly clustered data spaces but has less pruning power for scattered clusters. It is highly effective in specific data configurations. (Source Code: https://github.com/rajib1346/kNN/blob/main/idStar.py)
31. HC–O accelerates the candidate refining process but presumes a stable distribution of queries. Its effectiveness can be high but might vary significantly depending on the query distribution and dataset characteristics. (Source Code: https://github.com/rajib1346/kNN/blob/main/HC-O).

kNN join approach

Table 5 offers an insightful analysis, contrasting a range of kNN Join methodologies that are explored in this review, evaluating their effectiveness in various scenarios.

Table 5 Pros. and Cons of kNN Join

Techniques	Pros.	Cons.	Datasets and performances
Ball tree [100]	1. The Ball Tree kNN algorithm can effectively deal with high-dimensional data, making it a valuable instrument for applications such as image and speech recognition 2. It is faster than other brute-force algorithms for nearest-neighbor search, particularly for large data sets 3. The algorithm is adaptable and can accommodate a variety of distance metrics, making it appropriate for a wide range of data types	1. Creating a ball tree structure can take a significant amount of time, especially when dealing with datasets that are of considerable size 2. The algorithm is sensitive to the distance metric chosen, and its efficacy may suffer if the incorrect metric is selected 3. The algorithm may not always return the exact *k* Nearest Neighbors if distances between sites are tied	Dataset: Uniform dataset Construction time: 2301.80 ms
k-d tree [19]	1. The k-d tree structure enables efficient nearest neighbor searches, making it possible to perform such searches even on large datasets 2. The algorithm is straightforward and simple to implement 3. *O(nlogn)* is a reasonable time complexity for many applications and is quicker than brute-force search	1. The k-d tree structure can be inefficient for high-dimensional data, as the splitting process becomes less effective in higher dimensions 2. Building the k-d tree can be time-consuming for large datasets 3. The algorithm does not provide exact solutions for nearest neighbor search, but rather returns an approximation that may be suboptimal in some cases	Dataset: Signal dataset k = 16 Search time: 4.5 ms
PAT [101]	1. The PAT Algorithm is extraordinarily effective and can rapidly identify *k* nearest neighbors for enormous datasets 2. Using principal component analysis to construct the search tree enables the algorithm to identify the most pertinent characteristics of the dataset, thereby minimizing the impact of irrelevant characteristics on the search process 3. The algorithm's elimination criterion reduces the computational burden by discarding irrelevant search paths	1. The PAT Algorithm is extremely sensitive to the number of subsets and elimination threshold chosen. Choosing inappropriate values for these parameters can hinder the efficacy of the algorithm 2. The algorithm is only applicable to Euclidean distance measures, limiting its applicability in situations where other distance measures are more suitable 3. Due to the impact of the curse of dimensionality on the search process, the PAT Algorithm's performance may decrease considerably in high-dimensional datasets	Dataset: Image Lena k = 8192 Query times: 0.186 s Dataset: Image Boat Query times: 0.194 s Dataset: Image Babbon Query times: 0.385
MuX [7, 8]	Engineered to minimize I/O and CPU expenditures	Efficiency diminishes with increased dimensions; demands substantial memory resources	Dataset: Sequoia Total time: Between 500 and 1000 s
Gorder [94]	Limits random access and eliminates unproductive blocks	Requires extensive computational effort; tailored for static data	Dataset: Corel Image The average speed-up factor of Gorder over MuX is 0.59
iJoin [95]	Renowned for flexibility and dynamic adjustability	Incurs significant costs in dynamic data scenarios	Dataset: Uniform data CPU time: 1 h Dataset: KDD CPU time: 10 h

Table 5 (continued)

Techniques	Pros.	Cons.	Datasets and performances
IIB and IIIB [96]	Proficient in handling sparse datasets	Restricted adaptability for correlated data; room for performance enhancement	Dataset: Yeast and Worm I/O time: 3.5 h CPU time: Between 200 and 250 h
kNNJoin + [28]	Facilitates streamlined searches and dynamic modifications	Struggles with real-time criteria; hampered by distance calculation costs and node overlaps	Dataset: Synthetic dataset Elapsed time: 0 s (After dataset optimization)
HDR-tree [34]	Streamlines identification of affected users; adept at multi-dimensional data handling	Inability to accommodate deletions and batch updates	Dataset: NUS-WIDE Image dataset
EkNNJ [98]	Supports efficient, batch, and lazy updates; enhanced deletion options	Falls short in fully dynamic HD kNN Join support; opportunities for deletion optimization	Dataset: NUS-WIDE Image HDR Tree: 250 s HDR Forest: 250 s HDR + Forest: 250 s HDR*Forest: 100 s
H-BNLJ and H-BRJ [102]	Simple to implement	Lag in speed and scalability; elevated communication overhead	Dataset: OpenStreet Running time (H-BNJ): $10^3 to 10^4$ Running time (H-BRJ): Between $10^4 to 10^5$
PGBJ [12]	Communication overhead remains unaffected by increased k values; minimal disk usage	Struggles with high-dimensional data; performance heavily reliant on pivot selection; time-intensive for large datase	Dataset: Forest Running time: 1 s Shuffling cost: 4 Dataset: OSM Running time: 0.1 s Shuffling cost: 2 s

1. The Ball Tree kNN algorithm is valued for its competency in processing high-dimensional data and speed, especially compared to brute-force algorithms. Its adaptability, underscored by its compatibility with various distance metrics, broadens its application scope. However, its construction can be time-consuming, and the algorithm's performance is contingent on the appropriate selection of the distance metric. There is also a probability of not obtaining the exact nearest neighbors in some instances. To elevate its effectiveness, optimizing the tree construction process and enhancing result precision could be focal areas. (Source Code: https://github.com/rajib1346/kNN/blob/main/Ball-Tree.py).
2. The k-d tree is renowned for its efficient nearest neighbor searches, even in large datasets, owing to its intuitive structure and reasonable time complexity of O(nlogn). However, its effectiveness is hampered in higher dimensions, where the tree structure becomes less proficient. Additionally, constructing the k-d tree for substantial datasets can be time-intensive, and the algorithm often yields approximations rather than exact solutions for nearest neighbor searches. Enhancements in handling high-dimensional data and improving the accuracy of search results could augment the k-d tree's effectiveness. (Source Code: https://github.com/rajib1346/kNN/blob/main/k-d%20Tree).
3. The PAT algorithm stands out for its speed in identifying k nearest neighbors in massive datasets, attributed to the incorporation of principal component analysis and an effective elimination criterion. However, its sensitivity to parameter selection and restriction to Euclidean distance measures can be limiting. Its performance is also compromised in high-dimensional datasets due to the curse of dimensionality. Amplifying its effectiveness could involve broadening its adaptability to various distance measures and optimizing its performance in high-dimensional spaces. (Source Code: https://github.com/rajib1346/kNN/blob/main/PAT.py).
4. MuX is crafted to cut down on I/O and CPU expenses, making it a cost-efficient option. However, its performance is hindered with an increase in dimensions and it requires a significant amount of memory. Its effectiveness could potentially be enhanced by optimizing memory management and scalability for handling a broader spectrum of dimensional complexity. (Source Code: https://github.com/rajib1346/kNN/blob/main/MuX.py).
5. Gorder Gorder is adept at minimizing random access and eliminating non-beneficial blocks, enhancing data processing efficiency. Yet, it is computationally demanding and is constrained to static data, limiting its versatility. It is potent in specific scenarios but could gain from enhanced computational efficiency and adaptability to dynamic data. (Source Code: https://github.com/rajib1346/kNN/blob/main/Gorder.py).
6. iJoin is appreciated for its adaptability and dynamic nature, catering to varying data scenarios. However, it becomes substantially expensive for dynamic data. Balancing its adaptive attributes with cost efficiency, especially in fluctuating data environments, could amplify its effectiveness. (Source Code: https://github.com/rajib1346/kNN/blob/main/iJoin%20Series.py)
7. IIB and IIIB algorithms are tailored for sparse datasets, offering specific application advantages. However, their applicability wanes for correlated datasets, indicating a

need for versatility. Enhancements in adaptability could transform these algorithms into more universally applicable solutions. (Source Code: https://github.com/rajib1346/kNN/blob/main/IIB_IIIB.py)

8. kNNJoin + is distinguished by its efficient searching and dynamic update features. However, it struggles to meet real-time requirements and incurs high costs in distance computation and node overlap. Its effectiveness could potentially be amplified by optimizing real-time performance and computational efficiency. (Source Code: https://github.com/rajib1346/kNN/blob/main/kNNJoin%2B.py).
9. HDR-tree makes searches efficient and is adept at handling high-dimensional data. However, it lacks support for deletions and batch updates. Increasing its flexibility to accommodate these features could significantly bolster its utility and effectiveness in dynamic and evolving data environments. (Source Code: https://github.com/rajib1346/kNN/blob/main/HDR-Tree.py)
10. EkNNJ excels in dynamic, batch, and lazy updates and optimizes deletion processes. But, it does not support fully dynamic HD kNN Join, and there is room for improving deletion optimization. Enhancing these aspects could propel EkNNJ into a more comprehensive solution for diverse data needs. (Source Code: https://github.com/rajib1346/kNN/blob/main/EkNNJ.py)
11. H-BNLJ and H-BRJ are lauded for their ease of implementation but are criticized for their slower performance, limited scalability, and high communication overhead. Addressing these performance and efficiency bottlenecks could enhance their applicability across larger and more complex datasets. (Source Code: https://github.com/rajib1346/kNN/blob/main/H-BNLJ_H-BRJ.py)
12. PGBJ maintains communication overhead even as k values increase and ensures low disk usage. However, its efficiency drops with high-dimensional data, and pivot selection and large dataset handling are areas of concern. Optimizing these aspects can render PGBJ more effective and versatile in varied data landscapes. (Source Code: https://github.com/rajib1346/kNN/blob/main/PGBJ.py)

Discussion

This review paper aims to explore various perspectives and viewpoints on the application of k-Nearest Neighbors (kNN) algorithms in machine learning, focusing on key issues such as model interpretability, handling imbalanced datasets, optimal parameter selection, scalability, computational efficiency, and robustness to noisy data. By synthesizing insights from existing literature and scholarly discussions, this paper aims to provide a comprehensive understanding of the challenges and opportunities associated with kNN models in real-world applications.

Clarification on model interpretability

1. Concern: How does the kNN model's interpretability compare to other machine learning models, particularly in the context of feature importance and decision-making transparency?

Concept: Researchers have highlighted the challenge of interpreting kNN models, particularly in comparison to more transparent models such as decision trees or linear regression. While kNN models offer simplicity and ease of implementation, their interpretability is often limited due to the lack of explicit decision rules and reliance on local patterns.

2. Concern: Are there any established techniques or methodologies for enhancing the interpretability of kNN models, especially when dealing with high-dimensional datasets or complex classification tasks?

Concept: Some scholars advocate for the use of techniques such as local interpretable model-agnostic explanations (LIME) or SHAP (SHapley Additive exPlanations) values to elucidate the decision-making process of kNN models and provide insights into feature importance.

Handling imbalanced datasets

Concern 1: What are the common challenges faced when applying kNN to imbalanced datasets, and how do researchers typically address these challenges?

Concept: Imbalanced datasets pose a common challenge in machine learning, and kNN models are no exception. The majority voting mechanism of kNN can be biased towards the majority class, leading to suboptimal performance on minority classes.

Concern 2: Are there specific modifications or techniques tailored for kNN models to improve performance on imbalanced datasets, such as resampling methods or algorithm adjustments?

Concept: Techniques such as oversampling (e.g., SMOTE) or undersampling (e.g., random undersampling) are commonly employed to address class imbalance in kNN models. Moreover, modified distance metrics or instance weighting methods can be utilized to give more importance to minority samples during the classification process.

Optimal parameter selection

Concern 1: How do researchers determine the optimal value of the k parameter in kNN models, considering factors such as dataset characteristics, model complexity, and computational efficiency?

Concept: Determining the optimal value of the k parameter in kNN models is a crucial aspect of model performance. The choice of k significantly impacts the model's bias-variance tradeoff, with smaller values of k leading to higher model variance and potential overfitting.

Concern 2: Are there any systematic approaches or best practices for tuning hyperparameters in kNN models to achieve optimal performance and generalization across different datasets?

Concept: Cross-validation techniques, grid search, or model selection criteria such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion) are often used to identify the optimal value of k based on validation performance. Additionally, adaptive kNN algorithms dynamically adjust the value of k based on local data characteristics.

Scalability and computational efficiency

Concern 1: What are the limitations of kNN models in terms of scalability and computational efficiency, especially when dealing with large datasets or real-time applications?

Concept: Scalability and computational efficiency are important considerations when applying kNN models to large datasets or real-time applications. The computational complexity of kNN increases linearly with the size of the dataset *and the dimensionality of the feature space, making it less suitable for high-dimensional data.*

Concern 2: Have there been any recent advancements or techniques proposed to mitigate these limitations and improve the scalability of kNN models without compromising accuracy?

Concept: Approximate nearest neighbor algorithms such as k-d trees or ball trees are commonly used to accelerate nearest neighbor search and improve computational efficiency in kNN models. Additionally, parallelization techniques and distributed computing frameworks can be employed to scale kNN to large datasets.

Robustness to noisy data

Concern 1: How does the performance of kNN models degrade in the presence of noisy or irrelevant features, and what strategies are commonly employed to enhance robustness to noisy data?

Concept: Noisy or irrelevant features can adversely affect the performance of kNN models, leading to degraded accuracy and increased computational overhead. The distance-based nature of kNN makes it sensitive to noisy data points, as they can distort the local neighborhood structure and influence classification decisions.

Concern 2: Are there any novel approaches or preprocessing techniques specifically designed to preprocess noisy datasets before applying kNN models, and how effective are they in practice?

Concept: Feature selection methods, namely Extensive Feature Selector (EFS) [143], Local Feature Selection (LFS) [144], Brilliant Probabilistic Feature Selector (BPFS) [145], Ensemble Feature Selection [146], outlier detection techniques, or robust distance metrics (e.g., Mahalanobis distance) are often employed to mitigate the impact of noisy data on kNN models. Preprocessing steps such as data normalization or outlier removal can also improve the robustness of kNN to noisy features.

This review paper has provided a comprehensive overview of key issues and perspectives surrounding the application of kNN algorithms in machine learning. By addressing concerns related to model interpretability, handling imbalanced datasets, optimal parameter selection, scalability, computational efficiency, and robustness to noisy data, this paper aims to contribute to a deeper understanding of the strengths and limitations of kNN models in real-world applications. Further research and exploration are warranted to address emerging challenges and advance the capabilities of kNN algorithms in the era of big data and complex machine learning tasks.

Conclusion

In the rapidly evolving sphere of machine learning and data science, kNN querying techniques have steadily asserted their indispensability, particularly in the realm of high-dimensional spaces. In the pursuit of overcoming the inherent challenges of k-nearest

neighbor (kNN) searches, particularly in high-dimensional data spaces, recent advancements have presented a plethora of refined methodologies. Our survey has ventured deep into the corridors of exact kNN methodologies, addressing a notable gap in current literature by focusing on kNN Search and kNN Join techniques. As we delved into 31 distinct kNN Search methods and 12 kNN Join methods, our examination transcended mere descriptions, encompassing a rigorous comparative analysis, weighing the merits and limitations of each. Techniques such as VA+-file and EOTI offer promising approaches to approximating kNN searches, effectively balancing the trade-offs between search performance and space complexity. Innovations like Bregman distances and the Δ+-tree index have been tailored to address the inefficiencies in processing high-dimensional data and the computational burdens of traditional kNN algorithms. Significant strides have been made with the array-index method, showcasing remarkable efficiency in handling skewed and correlated datasets, and the ACDB method, which utilizes the triangle inequality principle to reduce CPU load significantly. iDistance-PS continues to play a pivotal role in kNN searches, with its effectiveness being amplified by various partitioning techniques. Emerging strategies such as the PL-Tree, M-kNN, and C-kNN demonstrate the ongoing evolution of kNN methodologies to support not only accuracy and efficiency but also to cater to the dynamic nature of data structures and distributions. Furthermore, the NFL and LNN algorithms highlight an innovative trajectory in kNN searches by integrating feature diversity and local proximity concepts, respectively. Overall, these modified methods collectively represent a leap forward in resolving the classic kNN problems, each contributing uniquely to the reduction of computational costs, improvement of query precision, and accommodation of the multi-faceted nature of high-dimensional data spaces. The convergence of these advancements signifies a transformative phase in kNN search methodologies, opening new avenues for real-time, scalable, and efficient data processing across various domains.

Challenges and future directions

The kNN method has significant performance issues despite its widespread use. The choice of k is one of these concerns. If k is too large, the neighborhood may contain an inordinate number of points from various classes, whereas if k is too small, the results may be susceptible to noise points. The kNN's single model approach limits its ability to effectively manage and interpret the complex information in large datasets, unlike ensemble models that combine multiple techniques for better results. This has motivated the need for a more robust, adaptive, and efficient version of the kNN algorithm that can effectively leverage the information-rich environment of big datasets while ensuring computational efficiency.

We can overcome these challenges by proposing the Region-Based Neighbors Searching Classification Algorithm. The Region-based Neighbor Searching Algorithm is designed to efficiently classify large-scale datasets. It operates by establishing a multitude of regions to determine the relationships between a new test point and existing data points. The number of regions to be generated is contingent upon the quantity of neighboring points identified within a specified radius of the test point. This method is effective in reducing both the time and space complexity associated with searching within extensive datasets. Our proposed idea encompasses two pivotal components. Firstly,

we will integrate a point-wise dynamic searching technique to meticulously explore all potential relationships between the new test sample and existing points, allocating distinct regions corresponding to specific radius values. Secondly, we will institute a comprehensive linkage with ensemble learning. This amalgamation facilitates the algorithm's decision-making process, enabling it to draw inferences based on the multifaceted, region-wise related behaviors of the test sample. The proposed algorithm is expresses as follows:

let we have a two datasets D_1, and D_2. Where D_1 denotes the trainset and D_2 denotes the test set.

$$D_1 = \{z^1(z_1, z_2, ..., z_n), ..., z^r(z_1, z_2, z_3, ..., z_n)\}$$

and

$$D_2 = \{x^1(x_1, x_2, ..., x_n), ..., x^p(x_1, x_2, ..., x_n)\}$$

where $D_1 > D_2$, r and p are the number of instances of train and test set respectively, and n represents the number of attributes. Firstly, pick a test data from D_1, and also take the value of ε where ε denotes the radius of a circular region. Secondly, create a region say R_1 at point x^1 for the given radius ε. Samples from the training dataset that fall within a distance equal to or less than the specified radius ε will be identified as neighbors $N_i(x^1_{[R_1]})$ of the test data within region R_1. This process is done by calculating the Euclidean distance (E_d) between test data and train samples as:

$$E_d(x^1, z^r) = \sqrt{\sum_{r=1}^{n} (x^1 - z^r)} \tag{1}$$

where,$N_i(x^1_{[R_1]}) = \{x : x \in z^r, \mathrm{E_d}(x^1, z^r) \leq \varepsilon\}$.

Next calculate the total weight of each class for according to the formula 2 and 3:

$$W_a = \sum \frac{1}{E_d(x^1, z^r)} \tag{2}$$

$$W_b = \sum \frac{1}{E_d(x^1, z^r)} \tag{3}$$

where a and b denotes the target classes. After Create a list that contains all the neighbors $N_i\left(x^1_{[R_1]}\right)$ of region R_1.

let, $C = [N_1, N_2, ..., N_i]$.where, $C = \{x : x \in N_i(x^1_{[R_1]})\} and\ \mathrm{E_d}(x^1, N_i) \leq \varepsilon$; $i = 1,2,3,.....,n$

Implementing a dynamic scanning technique involves sequentially traversing each neighbor stored in C. At each neighbor, a circular region with the same radius ε is considered, enabling the extraction of additional relations between the test point and the training points. The regions for neighbors $N_1, N_2, N_3, ..., N_i$ are denoted as $R_2, R_3, R_4, ..., R_i$, respectively. Create a list, denoted as R, which encompasses all regions, including R_1. In any region the Euclidean distance (E_d) between x^1 and its neighbors are less than or equal to the given radius (ε) i.e., $E_d(x^1, N_i(x^1_{[R_i]})) \leq \varepsilon$. Then the total weight of each class for

regions $R_2, R_3, R_4, ..., R_i$ is computed utilizing the aforementioned formulas (2) and (3). A weight-activation function is applied on each region to predict class of x^1 as follows:

$$y(R_i) = \begin{cases} a; W_a > W_b \\ b; Otherwise \end{cases} \tag{4}$$

Finally, integrate the outcomes of all regions and take the most frequently repeated class as a final target class using the below formula:

$$y = mode\big(y(R_1), y(R_2), y(R_3), ..., y(R_i)\big) \tag{5}$$

Continue the same Steps for the rest of the test samples $x^2, x^3, x^4, ..., x^n$.

This innovative methodology is tailored to enhance the sensitivity and specificity of the KNN algorithm, optimizing it for big data classification tasks. The aim is to not only improve the accuracy and reliability of classification results but also ensure that the algorithm is computationally efficient and scalable to handle large volumes of data characteristic of contemporary datasets.

The proposed idea has been guided by the following objectives:

1. Enhance Information Extraction: Develop and implement a Region-Based Neighbors Searching approach that facilitates exhaustive exploration and capture of relationships between new test samples and existing data points, overcoming the information extraction limitations of traditional KNN.
2. Optimize Computational Efficiency: Address the time and space complexity issues inherent in the traditional KNN, ensuring that the enhanced algorithm is capable of handling big data classification tasks efficiently.
3. Integrate Ensemble Learning: Establish a linkage with ensemble learning models to amplify the decision-making capability of the KNN algorithm, enabling it to derive insights from the complex, region-wise related behaviors of test samples.
4. Improve Classification Accuracy: Ensure that the refined KNN model classifies large-scale data with a significantly low error rate, marking a substantial improvement over its traditional counterpart in terms of accuracy and reliability.

Through these objectives, this research seeks to contribute a sophisticated, robust, and efficient tool to the repertoire of machine learning algorithms for big data classification, combining the foundational strengths of KNN with innovative enhancements to meet the demands of the contemporary data landscape.

Acknowledgements
We would like to extend our heartfelt appreciation to Deakin Cyber Research and Innovation Centre, Deakin University, Australia and UGC, Bangladesh (Grant ID: 37.01.0000.073.12.021.23.2849, 2022-23) for their support in this research.

Author contributions
Rajib Kumar Halder: Conceptualization, Resources, Data Curation, Methodology, Formal analysis, Editing, Validation. Mohammed Nasir Uddin: Investigation, Validation, Project administration, Review & Editing. Ashraf Uddin: Investigation, Formal analysis, Funding acquisition, Original Draft, Review & Editing. Sunil Aryal: Investigation, Formal analysis, Original Draft, Review & Editing. Ansam Khraisat: Investigation, Formal analysis, Review & Editing.

Funding
This work is financially supported by the Deakin Cyber Research and Innovation Centre, Deakin University, Australia.

Availability of data and materials
All datasets are available at the link below: https://github.com/rajib1346/kNN.git; and available from the corresponding author upon request. No datasets were generated or analysed during the current study.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 18 April 2024 Accepted: 23 July 2024
Published online: 11 August 2024

References

1. Wikipedia contributors. K-nearest neighbors algorithm. 2023. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.
2. Andoni A, Indyk P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), Berkeley, CA, USA, 2006. https://doi.org/10.1109/focs.2006.49.
3. Bawa M, Condie T, Ganesan P. LSH forest. In Proceedings of the 14th International Conference on World Wide Web (WWW '05). 2005. https://doi.org/10.1145/1060745.1060840
4. Lv Q, Josephson W, Wang Z, Charikar M, Li K. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07), 2007. 950–961. https://www.csd.uoc.gr/~hy561/Data/Papers/p950-lv.pdf.
5. Jeǵou H, Douze M, Schmid C. Product quantization for nearest neighbor search. IEEE Trans Pattern Anal Mach Intell. 2011;33(1):117–28. https://doi.org/10.1109/tpami.2010.57.
6. Wang Y, Pan Z, Li R. A new Cell-Level search based Non-Exhaustive Approximate Nearest Neighbor (ANN) search algorithm in the framework of product quantization. IEEE Access. 2019;7:37059–70. https://doi.org/10.1109/access.2019.2900730.
7. Böhm C, Krebs F. Supporting KDD applications by the K-Nearest Neighbor join. In Lecture Notes in Computer Science. 2003. pp. 504–516. https://doi.org/10.1007/978-3-540-45227-0_50.
8. Böhm C, Krebs F. The K-Nearest neighbour join: turbo charging the KDD process. Knowl Inf Syst. 2004;6(6):728–49. https://doi.org/10.1007/s10115-003-0122-9.
9. Algorithm AS 136: A K-Means Clustering Algorithm—百度学术. (n.d.). https://xueshu.baidu.com/usercenter/paper/show?paperid=2815fe2e7eaf7485735d130eac76d330.
10. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY. An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans Pattern Anal Mach Intell. 2002;24(7):881–92. https://doi.org/10.1109/tpami.2002.1017616.
11. Breunig M, Kriegel H, Ng RT, Sander J. LOF. Sigmod Record. 2000;29(2):93–104. https://doi.org/10.1145/335191.335388.
12. Lü W, Shen Y, Su C, Ooi BC. Efficient processing of k nearest neighbor joins using MapReduce. Proc VLDB Endowment. 2012;5(10):1016–27. https://doi.org/10.14778/2336664.2336674.
13. Dasarathy BV. Nearest neighbor (NN) norms: NN pattern classification techniques. 1991. http://ci.nii.ac.jp/ncid/BA19940413.
14. Zhang S, Li X, Zong M, Zhu X, Wang R. Efficient KNN classification with different numbers of nearest neighbors. IEEE Trans Neural Netw Learn Syst. 2018;29(5):1774–85. https://doi.org/10.1109/tnnls.2017.2673241.
15. Guttman A. R-Trees. Sigmod Record. 1984;14(2):47–57. https://doi.org/10.1145/971697.602266.
16. Beckmann N, Kriegel H, Schneider R, Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD '90). 1990. https://doi.org/10.1145/93597.98741.
17. Kamel I, Faloutsos C. Hilbert R-tree: An Improved R-tree using Fractals. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), 1994. 500–509. http://cis.temple.edu/~vasilis/Courses/CIS750/Papers/HilbertRtree-Kamel.pdf.
18. Arge L, De Berg M, Haverkort H, Yi K. The priority R-tree. ACM Trans Algorithms. 2008;4(1):1–30. https://doi.org/10.1145/1328911.1328920.
19. Sproull RF. Refinements to nearest-neighbor searching ink-dimensional trees. Algorithmica. 1991;6(1–6):579–89. https://doi.org/10.1007/bf01759061.
20. Fukunaga K, Narendra PM. A branch and bound algorithm for computing K-Nearest neighbors. IEEE Trans Comput. 1975;C–24(7):750–3. https://doi.org/10.1109/t-c.1975.224297.
21. (No date) Chapter 34 data structures and algorithms for nearest neighbor search ... Available at: http://algorithmics.lsi.upc.edu/docs/practicas/p311-yianilos.pdf (Accessed: 29 October 2023).
22. Bozkaya T, Ozsoyoglu M. Distance-based indexing for high-dimensional metric spaces. Sigmod Record. 1997;26(2):357–68. https://doi.org/10.1145/253262.253345.
23. Weber R, Schek H, Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998; 194–205.
24. Beyer K, Goldstein J, Ramakrishnan R, Shaft U. When is "Nearest Neighbor" meaningful? In Lecture Notes in Computer Science 1999. pp. 217–235. https://doi.org/10.1007/3-540-49257-7_15.

25. Kouiroukidis N, Evangelidis G. The Effects of Dimensionality Curse in High Dimensional kNN Search. 15th Panhellenic Conference on Informatics, Kastoria, Greece, 2011. https://doi.org/10.1109/pci.2011.45.
26. Cui B, Ooi BC, Su J, Tan K. Contorting high dimensional data for efficient main memory KNN processing. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03). 2003. https://doi.org/10.1145/872757.872815.
27. Garcia V, Debreuve É, Nielsen F, Barlaud M. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. 2010 IEEE International Conference on Image Processing, Hong Kong, China, 2010. https://doi.org/10.1109/icip.2010.5654017
28. Chen Y, Zhang R, Huang Y, Xiong H. High-dimensional kNN joins with incremental updates. GeoInformatica. 2009;14(1):55–82. https://doi.org/10.1007/s10707-009-0076-5.
29. Garcia V, Debreuve É, Barlaud M. Fast k nearest neighbor search using GPU. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Anchorage, AK, USA, 2008. https://doi.org/10.1109/cvprw.2008.4563100
30. Wold S, Esbensen KH, Geladi P. Principal component analysis. Chemom Intell Lab Syst. 1987;2(1–3):37–52. https://doi.org/10.1016/0169-7439(87)80084-9.
31. Chakrabarti K, Mehrotra S. Local dimensionality reduction: a new approach to indexing high dimensional spaces. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 89–100. 2000. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ldr.pdf.
32. Abdi H, Williams LJ. Principal component analysis. WIREs Comput Stat. 2010;2(4):433–59. https://doi.org/10.1002/wics.101.
33. Vidal R, Ma Y, Sastry SS. Principal component analysis. In Interdisciplinary applied mathematics. 2016. pp. 25–62. https://doi.org/10.1007/978-0-387-87811-9_2.
34. Yang C, Yu X, Yang L. Continuous KNN join processing for real-time recommendation. 2014 IEEE International Conference on Data Mining, Shenzhen, China, 2014. https://doi.org/10.1109/icdm.2014.20
35. Kibriya AM, Frank E. An empirical comparison of exact nearest neighbour algorithms. In Lecture Notes in Computer Science. 2007. pp. 140–151. https://doi.org/10.1007/978-3-540-74976-9_16.
36. Bhatia N. Survey of nearest neighbor techniques. 2010. arXiv.org. https://arxiv.org/abs/1007.0085.
37. RezaAbbasifard M, Ghahremani B, Naderi H. A survey on nearest neighbor search methods. Int J Comput Appl. 2014;95(25):39–52. https://doi.org/10.5120/16754-7073.
38. Liu T, Moore AW, Yang K, Gray AG. An investigation of practical approximate nearest neighbor algorithms. Neural Inf Proc Syst. 2004; 17: 825–832. http://papers.nips.cc/paper/2666-an-investigation-of-practical-approximate-nearest-neighbor-algorithms.pdf.
39. Li W, Zhang Y, Sun Y, Wang W, Li M, Zhang W, Lin X. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. IEEE Trans Knowl Data Eng. 2020;32(8):1475–88. https://doi.org/10.1109/tkde.2019.2909204.
40. Song G, Rochas J, Huet F, Magoulès F. Solutions for processing K Nearest neighbor joins for massive data on MapReduce. 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, Finland, 2015. https://doi.org/10.1109/pdp.2015.79
41. Song G, Rochas J, Beze LE, Huet F, Magoulès F. K Nearest neighbour joins for big data on MapReduce: a theoretical and experimental analysis. IEEE Trans Knowl Data Eng. 2016;28(9):2376–92. https://doi.org/10.1109/tkde.2016.2562627.
42. Chen Y, Ooi BC, Tan K, Jagadish HV. Indexing the Distance: An Efficient Method to KNN Processing. In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 421–430. 2001. https://www.vldb.org/conf/2001/P421.pdf.
43. Jagadish HV, Ooi BC, Tan K, Chen Y, Zhang R. IDistance. ACM Trans Database Syst. 2005;30(2):364–97. https://doi.org/10.1145/1071610.1071612.
44. Hu J, Cui B, Shen HT. Diagonal Ordering: a new approach to high-dimensional KNN processing. In Proceedings of the 15th Australasian Database Conference—Volume 27 (ADC '04). Australian Computer Society, Inc., AUS, 2004. 39–47. https://crpit.com/confpapers/CRPITV27Hu.pdf.
45. Olliffe IT. Principal component analysis: a beginner's guide—I. Introduction and application. Weather. 1990;45(10):375–82. https://doi.org/10.1002/j.1477-8696.1990.tb05558.x.
46. Berchtold S. The X-Tree : an index structure for High-Dimensional data. 2001. https://www.semanticscholar.org/paper/The-X-tree-%3A-An-Index-Structure-for-Data-Berchtold-Keim/774db16a3f25a73ceda9e6ab4d5a8b8f3c40605d.
47. Ferhatosmanoğlu H, Tuncel E, Agrawal D, Abbadi AE. High dimensional nearest neighbor searching. Inf Syst. 2006;31(6):512–40. https://doi.org/10.1016/j.is.2005.01.001.
48. Pan Y, Pan Z, Wang Y, Wang W. A new fast search algorithm for exact k-nearest neighbors based on optimal triangle-inequality-based check strategy. Knowl-Based Syst. 2020;189: 105088. https://doi.org/10.1016/j.knosys.2019.105088.
49. Almalawi AM, Fahad A, Tari Z, Cheema MA, Khalil I. k NNVWC: an efficient k -nearest neighbors approach based on various-widths clustering. IEEE Trans Knowl Data Eng. 2016;28(1):68–81. https://doi.org/10.1109/TKDE.2015.2460735.
50. Cayton L. Fast nearest neighbor retrieval for bregman divergences. In Proceedings of the 25th International Conference on Machine Learning (ICML '08). Association for Computing Machinery, New York, NY, USA. 2008. https://doi.org/10.1145/1390156.1390171.
51. Zhang Z, Ooi BC, Parthasarathy S, Tung AKH. Similarity search on Bregman divergence. Proc VLDB Endowment. 2009;2(1):13–24. https://doi.org/10.14778/1687627.1687630.
52. Song Y, Gu Y, Zhang R. BrePartition: Optimized High-Dimensional kNN Search with Bregman Distances. 2020. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2006.00227.

53. Aghbari ZA, Makinouchi A. Linearization approach for efficient KNN search of High-Dimensional Data. In Lecture Notes in Computer Science. 2004. pp. 229–238. https://doi.org/10.1007/978-3-540-27772-9_24.
54. Cui B, Coi BC, Su J, Tan K. Indexing high-dimensional data for efficient in-memory similarity search. IEEE Trans Knowl Data Eng. 2005;17(3):339–53. https://doi.org/10.1109/tkde.2005.46.
55. Hong H, Guo J, Wang B. An improved KNN algorithm based on adaptive cluster distance bounding for high dimensional indexing. 2012 Third Global Congress on Intelligent Systems, Wuhan, China. 2012. https://doi.org/10.1109/gcis.2012.86.
56. Schuh MA, Wylie T, Banda JM, Angryk RA. A comprehensive study of iDistance Partitioning Strategies for KNN Queries and High-Dimensional Data Indexing. In Lecture Notes in Computer Science. 2013. pp. 238–252. https://doi.org/10.1007/978-3-642-39467-6_22.
57. Zhang J, Zhou X, Wang W, Shi B, Pei J. Using high dimensional indexes to support relevance feedback based interactive images retrieval. In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06). VLDB Endowment, 2006. 1211–1214. https://doi.org/10.5555/1182635.1164246.
58. Shen HT, Ooi BC, Zhou X. Towards effective indexing for very large video sequence database. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05). Association for Computing Machinery, New York, NY, USA. 2005. https://doi.org/10.1145/1066157.1066240.
59. Ilarri S, Mena E, Illarramendi A. Location-dependent queries in mobile contexts: distributed processing using mobile agents. IEEE Trans Mob Comput. 2006;5(8):1029–43. https://doi.org/10.1109/tmc.2006.118.
60. Doulkeridis C, Vlachou A, Kotidis Y, Vazirgiannis M. Peer-to-peer similarity search in metric spaces. In Proceedings of the 33rd International Conference on Very Large Data Bases, 2007. 986–997. http://www.db-net.aueb.gr/files/2007VLDB.pdf.
61. Qu L, Chen Y, Xiao Y. iDistance Based Interactive Visual Surveillance Retrieval Algorithm. 2008 International Conference on Intelligent Computation Technology and Automation (ICICTA), Changsha, China, 2008. https://doi.org/10.1109/icicta.2008.13.
62. Wang J, Lu J, Zheng F, Ge T, Chen C. PL-Tree: an efficient indexing method for high-dimensional data. In Lecture Notes in Computer Science. 2013. pp. 183–200. https://doi.org/10.1007/978-3-642-40235-7_11.
63. Schuh MA, Wylie T, Angryk RA. Mitigating the Curse of Dimensionality for Exact kNN Retrieval. In the Twenty-Seventh International Flairs Conference. 2014. https://dblp.uni-trier.de/db/conf/flairs/flairs2014.html#SchuhWA14.
64. Schuh MA, Wylie T, Angryk RA. Improving the Performance of High-Dimensional kNN Retrieval through Localized Dataspace Segmentation and Hybrid Indexing. In Lecture Notes in Computer Science. 2013. pp. 344–357. https://doi.org/10.1007/978-3-642-40683-6_26.
65. Wylie T, Schuh MA, Sheppard JW, Angryk RA. Cluster Analysis for Optimal Indexing. In FLAIRS Conference. 2013. https://academic.timwylie.com/files/Wylie_2013_FLAIRS.pdf.
66. Tang B, Yiu ML, Hua KA. Exploit every bit: Effective caching for high-dimensional nearest neighbor search (extended abstract). 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA. 2017. https://doi.org/10.1109/icde.2017.29.
67. Shang X, Zhu Z, Leimkuhler B, Storkey A. Neural Information Processing Systems (NIPS). Learning to Prune in Metric and Non-Metric Spaces. NIPS: Neural Information Processing Systems. 2015. https://www.research.ed.ac.uk/portal/en/publications/covariancecontrolled-adaptive-langevin-thermostat-for-largescale-bayesian-sampling(765825cc-13d9-40d1-8a9e-e696a60e9e36).html.
68. Weber R, Blott SM. An Approximation-Based Data Structure for Similarity Search. ResearchGate. 1998. https://www.researchgate.net/publication/2577157_An_Approximation-Based_Data_Structure_for_Similarity_Search.
69. Cover TM, Hart PD. Nearest neighbor pattern classification. IEEE Trans Inf Theory. 1967;13(1):21–7. https://doi.org/10.1109/tit.1967.1053964.
70. Bailey TL, Jain A. A note on Distance-Weighted K-Nearest Neighbor rules. IEEE Trans Syst Man Cybern. 1978;8(4):311–3. https://doi.org/10.1109/tsmc.1978.4309958.
71. Gowda KC, Krishna G. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (Corresp.). IEEE Trans Inf Theory. 1979;25(4):488–90. https://doi.org/10.1109/tit.1979.1056066.
72. Gates GW. The reduced nearest neighbor rule (Corresp.). IEEE Trans Inf Theory. 1972;18(3):431–3. https://doi.org/10.1109/tit.1972.1054809.
73. Viadinugroho RAA. Imbalanced Classification in Python: SMOTE-ENN Method. Medium. 2022. https://towardsdatascience.com/imbalanced-classification-in-python-smote-enn-method-db5db06b8d50.
74. Ritter GL, Woodruff HB, Lowry SR, Isenhour TL. An algorithm for a selective nearest neighbor decision rule (Corresp.). IEEE Trans Inf Theory. 1975;21(6):665–9. https://doi.org/10.1109/tit.1975.1055464.
75. Kolahdouzan MR, Shahabi C. Voronoi-Based K Nearest neighbor search for spatial network databases. In Elsevier eBooks. 2004. pp. 840–851. https://doi.org/10.1016/b978-012088469-8.50074-7.
76. Guo G, Wang H, Bell DA, Bi Y, Greer K. KNN model-based approach in classification. In Lecture Notes in Computer Science. 2003. pp. 986–996. https://doi.org/10.1007/978-3-540-39964-3_62.
77. Chiang T, Lo H, Lin S. A ranking-based KNN approach for Multi-Label classification. J Mach Learn Res. 2012; 81–96. http://proceedings.mlr.press/v25/chiang12/chiang12.pdf.
78. Yong Z, Li Y, Xia S. An improved KNN text classification algorithm based on clustering. J Comput. 2009. https://doi.org/10.4304/jcp.4.3.230-237.
79. Li S, Chan KL, Wang C. Performance evaluation of the nearest feature line method in image classification and retrieval. IEEE Trans Pattern Anal Mach Intell. 2000;22(11):1335–9. https://doi.org/10.1109/34.888719.
80. Zheng W, Zhao L, Zou C. Locally nearest neighbor classifiers for pattern classification. Pattern Recogn. 2004;37(6):1307–9. https://doi.org/10.1016/j.patcog.2003.11.004.
81. Kuang Q, Zhao L. A practical GPU based kNN algorithm. International Symposium on Computer Science and Computational Technology (ISCSCT). 2009.
82. Batcher KE. Sorting networks and their applications. In Proceedings of the April 30--May 2, 1968, Spring Joint Computer Conference (AFIPS '68 (Spring)). 1968. https://doi.org/10.1145/1468075.1468121.

83. Sismanis N, Pitsianis N, Sun X. Parallel search of k-nearest neighbors with synchronous operations. 2012 IEEE Conference on High Performance Extreme Computing, Waltham, MA, USA, 2012. https://doi.org/10.1109/hpec.2012.6408667.
84. Liu B, Lee W, Lee DL. Supporting Complex Multi-Dimensional Queries in P2P Systems. 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), Columbus, OH, USA, 2005. https://doi.org/10.1109/icdcs.2005.75.
85. Li M, Lee WC, Sivasubramaniam A, Zhao J. Supporting K nearest neighbors query on high-dimensional data in P2P systems. Front Comp Sci. 2008;2(3):234–47. https://doi.org/10.1007/s11704-008-0026-7.
86. Jagadish HV, Ooi BC, Vu QH, Zhang R, Zhou A. VBI-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes. 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 2006. https://doi.org/10.1109/icde.2006.169.
87. Qiao B, Ding L, Wei Y, Wang X. A KNN Query Processing Algorithm over High-Dimensional Data Objects in P2P Systems. In Advances in intelligent and soft computing. 2012. pp. 133–139. https://doi.org/10.1007/978-3-642-28314-7_19.
88. Jian L, Wang C, Liu Y, Liang S, Yi W, Shi Y. Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). J Supercomput. 2011;64(3):942–67. https://doi.org/10.1007/s11227-011-0672-7.
89. Xiao B, Biros G. Parallel algorithms for nearest neighbor search problems in high dimensions. SIAM J Sci Comput. 2016;38(5):S667–99. https://doi.org/10.1137/15m1026377.
90. Clarke LJ, Glendinning I, Hempel R. The MPI Message Passing Interface Standard. In Birkhäuser Basel eBooks. 1994. pp. 213–218. https://doi.org/10.1007/978-3-0348-8534-8_21.
91. Dagum L, Menon R. OpenMP: an industry standard API for shared-memory programming. IEEE Comput Sci Eng. 1998;5(1):46–55. https://doi.org/10.1109/99.660313.
92. Muhr D, Affenzeller M. Hybrid (CPU/GPU) exact nearest neighbors search in High-Dimensional Spaces. In IFIP advances in information and communication technology. 2022. pp. 112–123. https://doi.org/10.1007/978-3-031-08337-2_10.
93. Luebke D, Harris MJ, Govindaraju NK, Lefohn A, Houston MJ, Owens JD, Segal MN, Papakipos M, Buck I. S07---GPGPU. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06). Association for Computing Machinery, New York, NY, USA. 2006. https://doi.org/10.1145/1188455.1188672.
94. Xia C, Lu H, Ooi B, Hu J. GORDERAn Efficient Method for KNN join processing. In Elsevier eBooks. 2004. pp. 756–767. https://doi.org/10.1016/b978-012088469-8/50067-x.
95. Chen Y, Cui B, Wang S, Su J. Efficient index-based KNN join processing for high-dimensional data. Inf Softw Technol. 2007;49(4):332–44. https://doi.org/10.1016/j.infsof.2006.05.006.
96. Wang J. Efficient K-Nearest Neighbor join algorithms for high dimensional sparse data. 2010. arXiv.org. https://arxiv.org/abs/1011.2807.
97. Achlioptas D. Database-friendly random projections. Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2001. https://doi.org/10.1145/375551.375608.
98. Ukey N, Yang Z, Zhang G, Liu B, Li B, Zhang W. Efficient kNN join over dynamic high-dimensional data. In Lecture Notes in Computer Science. 2022. pp. 63–75. https://doi.org/10.1007/978-3-031-15512-3_5.
99. Dean JM, Ghemawat S. MapReduce. Commun ACM. 2008;51(1):107–13. https://doi.org/10.1145/1327452.1327492.
100. CiteSeerX. (n.d.). *CiteSeerX*. https://citeseerx.ist.psu.edu/doc_view/pid/17ac002939f8e950ffb32ec4dc8e86bdd8cb5ff1#citations.
101. McNames J. A fast nearest-neighbor algorithm based on a principal axis search tree. IEEE Trans Pattern Anal Mach Intell. 2001;23(9):964–76. https://doi.org/10.1109/34.955110.
102. Zhang C, Li F, Jestes J. Efficient parallel kNN joins for large data in MapReduce. ACM Int Conf Proc Ser. 2012. https://doi.org/10.1145/2247596.2247602.
103. Pan Z, Wang Y, Ku W. A new k-harmonic nearest neighbor classifier based on the multi-local means. Expert Syst Appl. 2017;67:115–25. https://doi.org/10.1016/j.eswa.2016.09.031.
104. Pan Z, Wang Y, Ku W. A new general nearest neighbor classification based on the mutual neighborhood information. Knowl-Based Syst. 2017;121:142–52. https://doi.org/10.1016/j.knosys.2017.01.021.
105. De Figueiredo JJS, Oliveira F, Esmi E, Freitas L, Schleicher J, Novais A, Sussner P, Green S. Automatic detection and imaging of diffraction points using pattern recognition. Geophys Prospect. 2012;61(s1):368–79. https://doi.org/10.1111/j.1365-2478.2012.01123.x.
106. Nguyen B, Morell C, De Baets B. Large-scale distance metric learning for k-nearest neighbors regression. Neurocomputing. 2016;214:805–14. https://doi.org/10.1016/j.neucom.2016.07.005.
107. Song Y, Liang J, Lü J, Zhao X. An efficient instance selection algorithm for k nearest neighbor regression. Neurocomputing. 2017;251:26–34. https://doi.org/10.1016/j.neucom.2017.04.018.
108. Stone CJ. Consistent nonparametric regression. Ann Stat. 1977. https://doi.org/10.1214/aos/1176343886.
109. Angiulli F, Basta S, Pizzuti C. Distance-based detection and prediction of outliers. IEEE Trans Knowl Data Eng. 2006;18(2):145–60. https://doi.org/10.1109/tkde.2006.29.
110. Ghoting A, Parthasarathy S, Otey ME. Fast mining of distance-based outliers in high-dimensional datasets. Data Min Knowl Disc. 2008;16(3):349–64. https://doi.org/10.1007/s10618-008-0093-2.
111. Jin N, Chen L, Zhou C, Wen Y. Parameter k search strategy in outlier detection. Pattern Recogn Lett. 2018;112:56–62. https://doi.org/10.1016/j.patrec.2018.06.007.
112. Ramaswamy S, Rastogi R, Shim K. Efficient algorithms for mining outliers from large data sets. Sigmod Record. 2000;29(2):427–38. https://doi.org/10.1145/335191.335437.
113. Jiang S, Pang G, Wu M, Kuang L. An improved K-nearest-neighbor algorithm for text categorization. Expert Syst Appl. 2012;39(1):1503–9. https://doi.org/10.1016/j.eswa.2011.08.040.
114. Cavalcante HG. A question classification in closed domain question-answer systems. Int J Appl Inf Syst (IJAIS). 2021;12:1–5. https://doi.org/10.5120/ijais2021451913.

115. Bijalwan V, Kumar V, Kumari P, Pascual J. KNN based machine learning approach for text and document mining. Int J Database Theory Appl. 2014;7(1):61–70. https://doi.org/10.14257/ijdta.2014.7.1.06.
116. Zhao J, Han J, Shao L. Unconstrained face recognition using a Set-to-Set distance measure on deep learned features. IEEE Trans Circuits Syst Video Technol. 2018;28(10):2679–89. https://doi.org/10.1109/tcsvt.2017.2710120.
117. Tofighi A, Khairdoost N, Monadjemi SA, Jamshidi K. A robust face recognition system in image and video. Int J Image Graphics Signal Proc. 2014;6(8):1–11. https://doi.org/10.5815/ijigsp.2014.08.01.
118. Zhang J, Yin Z, Chen P, Nichele S. Emotion recognition using multi-modal data and machine learning techniques: a tutorial and review. Inf Fusion. 2020;59:103–26. https://doi.org/10.1016/j.inffus.2020.01.011.
119. Murugappan M. Human emotion classification using wavelet transform and KNN. 2011 International Conference on Pattern Analysis and Intelligence Robotics, Kuala Lumpur, Malaysia, 2011. https://doi.org/10.1109/icpair.2011.5976886.
120. Guru DS, Sharath YH, Manjunath S. Texture features and KNN in classification of flower images. Int J Comput Appl. 2010;1:21–9.
121. Zawbaa HM, Abbass M, Hazman M, Hassenian AE. Automatic fruit image recognition system based on shape and color features. In Communications in computer and information science. 2014. pp. 278–290. https://doi.org/10.1007/978-3-319-13461-1_27.
122. Zanchettin C, Bezerra BLD, Azevedo WW. A KNN-SVM hybrid model for cursive handwriting recognition. The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, QLD, Australia, 2012. 2012. https://doi.org/10.1109/ijcnn.2012.6252719.
123. Hamid NA, Sjarif NNA. Handwritten recognition using SVM, KNN and neural network. arXiv (Cornell University). 2017. https://arxiv.org/pdf/1702.00723.pdf.
124. Akila S, Reddy US. Cost-sensitive Risk Induced Bayesian Inference Bagging (RIBIB) for credit card fraud detection. J Comput Sci. 2018;27:247–54. https://doi.org/10.1016/j.jocs.2018.06.009.
125. Imandoust SB, Bolandraftar M. Application of K-nearest neighbor (KNN) approach for predicting economic events theoretical background. Int J Eng Res Appl. 2013;3:605–10.
126. Zheng B, Zheng K, Xiao X, Su H, Yin H, Zhou X, Li G. Keyword-aware continuous kNN query on road networks. 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 2016. 2016. https://doi.org/10.1109/icde.2016.7498297.
127. Tripathy DP, Parida S, Khandu L. Safety risk assessment and risk prediction in underground coal mines using machine learning techniques. J Inst Eng India Series D. 2021;102(2):495–504. https://doi.org/10.1007/s40033-021-00290-1.
128. Mohsen S, Elkaseer A, Scholz S. Human activity recognition using K-Nearest Neighbor Machine Learning Algorithm. In Smart innovation, systems and technologies. 2021. pp. 304–313. https://doi.org/10.1007/978-981-16-6128-0_29.
129. Selma C, Haouzi HBE, Thomas P, Gaudreault J, Morin M. An iterative closest point method for measuring the level of similarity of 3D log scans in wood industry. In Studies in computational intelligence 2018. pp. 433–444. https://doi.org/10.1007/978-3-319-73751-5_33.
130. Chabanet S, Thomas P, El-Haouzi HB, Morin M, Gaudreault J. A kNN approach based on ICP metrics for 3D scans matching: an application to the sawing process. IFAC-PapersOnLine. 2021;54(1):396–401. https://doi.org/10.1016/j.ifacol.2021.08.045.
131. Al-Faiz MZ, Ali AA, Miry AH. A K-Nearest Neighbor based algorithm for human arm movements recognition using EMG signals. Al-mağallaẗ Al-ʻirāqiyyaẗ Al-handasaẗ Al-kahrabāʾiyyaẗ Wa-al-ilikttrūniyyaẗ. 2010;6(2): 158–166. https://doi.org/10.33762/eeej.2010.54888.
132. Shen B, Zhao Y, Li G, Zheng W, Qin Y, Yuan B, Rao Y. V-Tree: Efficient kNN Search on Moving Objects with Road-Network Constraints. 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 2017. 2017. https://doi.org/10.1109/icde.2017.115.
133. Fiorini L, Mancioppi G, Semeraro F, Fujita H, Cavallo F. Unsupervised emotional state classification through physiological parameters for social robotics applications. Knowl-Based Syst. 2020;190: 105217. https://doi.org/10.1016/j.knosys.2019.105217.
134. Markom MA, Adom AH, Shukor SAA, Rahim NA, Tan ESMM, Ilias B. Improved KNN scan matching for local map classification in mobile Robot Localisation application. IOP Conf Ser Mater Sci Eng. 2019;557(1):012019. https://doi.org/10.1088/1757-899x/557/1/012019.
135. Pinto AM, Rocha LF, Moreira AP. Object recognition using laser range finder and machine learning techniques. Robot Comput-Integr Manuf. 2013;29(1):12–22. https://doi.org/10.1016/j.rcim.2012.06.002.
136. Xu G, Pang Y, Bai Z, Wang Y, Lü Z. A fast point clouds registration algorithm for laser scanners. Appl Sci. 2021;11(8):3426. https://doi.org/10.3390/app11083426.
137. Li W, Yi P, Wu Y, Pan L, Li J. A new intrusion detection system based on KNN classification algorithm in wireless sensor network. J Electric Comput Eng. 2014;2014:1–8. https://doi.org/10.1155/2014/240217.
138. Liu G, Zhao H, Fan F, Liu G, Xu Q, Nazir S. An enhanced intrusion detection model based on improved KNN in WSNs. Sensors. 2022;22(4):1407. https://doi.org/10.3390/s22041407.
139. Yang J, Sun Z, Chen Y. Fault detection using the Clustering-KNN rule for gas sensor arrays. Sensors. 2016;16(12):2069. https://doi.org/10.3390/s16122069.
140. Zhou C, Tham C. GraphEL: A Graph-Based Ensemble Learning Method for Distributed Diagnostics and Prognostics in the Industrial Internet of Things. 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 2018. 2018. https://doi.org/10.1109/padsw.2018.8644943.
141. Liang S, Ning Y, Li H, Wang L, Mei Z, Ma Y, Zhao G. Feature selection and predictors of falls with foot force sensors using KNN-based algorithms. Sensors. 2015;15(11):29393–407. https://doi.org/10.3390/s151129393.
142. Dziubany M, Machhamer R, Laux H, Schmeink A, Gollmer KU, Burger G, Dartmann G. Machine Learning Based Indoor Localization Using a Representative k-Nearest-Neighbor Classifier on a Low-Cost IoT-Hardware. 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 2018. 2018. https://doi.org/10.23919/eusipco.2018.8553155.

143. Parlak B, Uysal AK. A novel filter feature selection method for text classification: extensive feature selector. J Inf Sci. 2023;49(1):59–78. https://doi.org/10.1177/0165551521991037.
144. Parlak B, Uysal AK. The effects of globalisation techniques on feature selection for text classification. J Inf Sci. 2021;47(6):727–39. https://doi.org/10.1177/0165551520930897.
145. Parlak B. A novel feature ranking algorithm for text classification: brilliant probabilistic feature selector (BPFS). Comput Intell. 2023;39(5):900–26. https://doi.org/10.1111/coin.12599.
146. Parlak B. Ensemble feature selection for single-label text classification: a comprehensive analytical study. Neural Comput Appl. 2023;35:19235–51. https://doi.org/10.1007/s00521-023-08763-y.
147. Mladenova T, Valova I. Comparative analysis between the traditional K-Nearest Neighbor and Modifications with Weight-Calculation, 2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 2022, pp. 961–965, https://doi.org/10.1109/ISMSIT56059.2022.9932693.
148. Briliani A, Irawan B, Setianingsih C. Hate Speech Detection in Indonesian Language on Instagram Comment Section Using K-Nearest Neighbor Classification Method, 2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS), Bali, Indonesia, 2019, pp. 98–104, https://doi.org/10.1109/IoTaIS47347.2019.8980398.

Publisher's Note