SCALING kNN QUERIES USING STATISTICAL LEARNING

ATOSHUM SAMUEL CAHSAI

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE
COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

09 2018

Abstract

The k-Nearest Neighbour (kNN) method is a fundamental building block for many sophisticated statistical learning models and has a wide application in different fields; for instance, in kNN regression, kNN classification, multi-dimensional items search, location-based services, spatial analytics, etc. However, nowadays with the unprecedented spread of data generated by computing and communicating devices has resulted in a plethora of low-dimensional large-scale datasets and their users' community, the need for efficient and scalable kNN processing is pressing. To this end, several parallel and distributed approaches and methodologies for processing exact kNN in low-dimensional large-scale datasets have been proposed; for example Hadoop-MapReduce-based kNN query processing approaches such as Spatial-Hadoop (SHadoop), and Spark-based approaches like Simba. This thesis contributes with a variety of methodologies for kNN query processing based on statistical and machine learning techniques over large-scale datasets.

This study investigates the exact kNN query performance behaviour of the well-known Big Data Systems, SHadoop and Simba, that proposes building multi-dimensional Global and Local Indexes over low dimensional large-scale datasets. The rationale behind such methods is that when executing exact kNN query, the Global and Local indexes access a small subset of a large-scale dataset stored in a distributed file system. The Global Index is used to prune out irrelevant subsets of the dataset; while the multiple distributed Local Indexes are used to prune out unnecessary data elements of a partition (subset).

The kNN execution algorithm of SHadoop and Simba involves loading data elements that reside in the relevant partitions from disks/network points to memory. This leads to significantly high kNN query response times; so, such methods are not suitable for low-latency applications and services. An extensive literature review showed that not enough attention has been given to access relatively small-sized but relevant data using kNN query only. Based on this limitation, departing from the traditional kNN query processing methods, this thesis contributes two novel solutions: Coordinator With Index (COWI) and Coordinator with No Index(CONI) approaches. The essence of both approaches rests on adopting a coordinator-based distributed processing algorithm and a way to structure computation and index the stored datasets that ensures that only a very small number of pieces of data are retrieved from the underlying data centres, communicated over the network, and processed by the coordinator for every kNN query. The expected outcome is that scalability is ensured and kNN queries can be processed in just tens of milliseconds. Both approaches are implemented using a NoSQL Database (HBase) achieving up to three orders of magnitude of performance gain compared with state of the art methods -SHadoop and Simba.

It is common practice that the current state-of-the-art approaches for exact kNN query processing in low-dimensional space use Tree-based multi-dimensional Indexing methods to

prune out irrelevant data during query processing. However, as data sizes continue to increase, (nowadays it is not uncommon to reach several Petabytes), the storage cost of Tree-based Index methods becomes exceptionally high, especially when opted to partition a dataset into smaller chunks. In this context, this thesis contributes with a novel perspective on how to organise low-dimensional large-scale datasets based on data space transformations deriving a Space Transformation Organisation Structure (STOS). STOS facilitates kNN query processing as if underlying datasets were uniformly distributed in the space. Such an approach bears significant advantages: first, STOS enjoys a minute memory footprint that is many orders of magnitude smaller than Index-based approaches found in the literature. Second, the required memory for such meta-data information over large-scale datasets, unlike related work, increases very slowly with dataset size. Hence, STOS enjoys significantly higher scalability. Third, STOS is relatively efficient to compute, outperforming traditional multivariate Index building times, and comparable, if not better, query response times.

In the literature, the exact kNN query in a large-scale dataset was limited to low-dimensional space; this is because the query response time and memory space requirement of the Tree-based index methods increase with dimension. Unable to solve such exponential dependency on the dimension, researchers assume that no efficient solution exists and propose approximation kNN in high dimensional space. Unlike the approximated kNN query that tries to retrieve approximated nearest neighbours from large-scale datasets, in this thesis a new type of kNN query referred to as estimated kNN query is proposed. The estimated kNN query processing methodology attempts to estimate the nearest neighbours based on the marginal cumulative distribution of underlying data using statistical copulas. This thesis showcases the performance trade-off of exact kNN and the estimate kNN queries in terms of estimation error and scalability. In contrast, kNN regression predicts that a value of a target variable based on kNN; but, particularly in a high dimensional large-scale dataset, a query response time of kNN regression, can be a significantly high due to the curse of dimensionality. In an effort to tackle this issue, a new probabilistic kNN regression method is proposed. The proposed method statistically predicts the values of a target variable of kNN without computing distance.

In different contexts, a kNN as missing value algorithm in high dimensional space in Pytha, a distributed/parallel missing value imputation framework, is investigated. In Pythia, a different way of indexing a high-dimensional large-scale dataset is proposed by the group (not the work of the author of this thesis); by using such indexing methods, scaling-out of kNN in high dimensional space was ensured. Pythia uses Adaptive Resonance Theory (ART) -a machine learning clustering algorithm- for building a data digest (aka signatures) of large-scale datasets distributed across several data machines. The major idea is that given an input vector, Pythia predicts the most relevant data centres to get involved in processing, for example, kNN. Pythia does not retrieve exact kNN. To this end, instead of accessing the entire

dataset that resides in a data-node, in this thesis, accessing only relevant clusters that reside in appropriate data-nodes is proposed. As we shall see later, such method has comparable accuracy to that of the original design of Pythia but has lower imputation time. Moreover, the imputation time does not significantly grow with a size of a dataset that resides in a data node or with the number of data nodes in Pythia. Furthermore, as Pythia depends utterly on the data digest built by ART to predict relevant data centres, in this thesis, the performance of Pythia is investigated by comparing different signatures constructed by a different clustering algorithms, the Self-Organising Maps.

In this thesis, the performance advantages of the proposed approaches via extensive experimentation with multi-dimensional real and synthetic datasets of different sizes and context are substantiated and quantified.

Acknowledgments

I would like to express my sincere appreciation and gratitude to many people who so generously contributed to the work presented in this thesis. Special mention goes to my enthusiastic supervisors, Peter Triantafillou and Christos Anagnostopoulos. Similar, profound gratitude goes to Nikos Ntarmos. My PhD has been a wonderful experience, and I thank Peter, Christos and Nikos wholeheartedly, for their academic guidance and enthusiastic encouragement throughout the research. They stimulated and directed me in publishing conference and journal papers. During these projects, they were always there to provide necessary assistance and academic resources. Without their support, I could not have done what I was able to do. I have very fond memories of my time working with them.

I am also hugely appreciative to Vincent Macaulay, especially for sharing his professional knowledge so willingly.

Special mention goes to Fotis Savva and Wei Ma for their feedback and support.

Finally, but by no means least, lots of thanks go to my family, particularly to my little daughter Christianne for their almost unbelievable support. However, my highest appreciation goes to my late wife Lili, who sacrificed dearly and through thick and thin stood by my side. Sadly, she did not live to celebrate the final submission of my thesis. She was the most important person in my world, and I dedicate this thesis to her loving memory.

Dedicated To my little daughter Christianne and in loving memory of my beloved wife Lili.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Rationale

The k-Nearest Neighbours (kNN) algorithm is a *lazy*-learner and a non-parametric method. It is one of the most straightforward Machine Learning (ML) algorithms and yet has robust application across many research and application fields. kNN is used to search for the k most similar elements to a given item within a finite collection. Often an item (element) is represented by a multi-dimensional vector, and hence similarity between vectors is defined by an appropriate distance metric, for instance, Euclidean distance, Manhattan distance, Mahalanobis distance, cosine similarity, etc.

The kNN has been widely used in academy and industry. For example kNN has been attracted a vast research community such as [5, 35, 63, 44, 91, 7, 87]. In industry, kNN has been widely used for concept search (documents containing similar topics), e.g. in lawsuit companies; recommender systems (recommending advertisement to display to a user); in face recognition (Herta Security) [1] using Deep Learning generates features of vectors that represent people's faces and adopts kNN for matching similar faces; in predicting economic events, e.g., predicting companies' financial distress and many more.

Due its relatively easy and straightforward implementation, easy interpretation and good performance, the kNN method is considered- by some researchers- as one of the top-10 Data Mining algorithms [85]. When kNN is used for classification, despite its simplicity, it is surprisingly versatile and works incredibly well in many applications [85]; [28] showed that under certain assumptions, the error of the *nearest neighbour* rule is bounded above by twice the Bayes error. kNN is particularly well suited for multi-modal classes as well as applications in which an item can have many class labels; for the assignment of functions to genes based on expression profiles.

[1] http://www.hertasecurity.com/en

Traditionally, the kNN algorithm was designed to run in a centralised computation system. To retrieve the top k-NN from a collection (dataset), the kNN algorithm scans the *whole dataset* sequentially. However, nowadays, as the unprecedented spread of data generated by computing and communicating devices has resulted in a plethora of large-scale datasets, sequentially scanning such massive dataset becomes infeasible as the kNN query response time might take hours, if not days.

In general, the ever increasing datasets impose a significant limitation on *scalability* - coping and performing well under the expanding data- of the traditional off-the-shelf tools and Data Mining algorithms. To tackle this problem, a body of research has focused on parallel & distributed Big Data systems such as Hadoop/MapReduce [31] and Spark [92] . In distributed & parallel systems, a large-scale dataset is stored in a distributed file system (DFS) in a cluster of commodity hardware. When a kNN algorithm particularly runs in a vanilla Hadoop/MapReduce or Spark, it benefits from the *intra-query parallelism*, i.e. scalability of the kNN query significantly improves as a result of parallel execution of the query over many machines of the cluster. The more machines added to the cluster, the higher the scalability gain is achieved. Nonetheless, in spite of the intra-query parallelism, executing kNN queries using the vanilla Hadoop/MapReduce or Spark has two main drawbacks as the whole dataset is scanned even in parallel:

- it has high query response time;
- it wastes precious resources by being occupied all machines that *do not contributing* to the final result.

For example, assume that all top k-NN data points to a query reside in one machine of the cluster. Then, during the query execution, accessing the entire dataset, including parts of the dataset that reside in the other machines that do not contain the answer, has nothing to contribute to the final result. It only wastes precious resources that would have otherwise been used for processing the next kNN queries in a queue, particularly in a streaming environment where the queries are not batch-able. Consequently, the vanilla Hadoop/MapReduce and Spark do not support inter-query parallelism: cannot support execution of more than one kNN queries at a given time, thus degrading the throughput of the decentralised and parallel system.

To this end, scaling-out exact kNN query processing, by indexing low-dimensional large-scale datasets that reside in a DFS, has recently attracted many researchers [5, 35, 7, 63, 45, 44, 87, 91]. During the index building process, a dataset is divided into several small-sized subsets (partitions); and hence, irrespective of the size of the whole dataset, at query time, only small but relevant data partitions are retrieved and processed. In the perspective of query scalability and performance, such a design has been proven superior compared with systems

that access the entire dataset (even in parallel). But, in spite of the incredible advantages, even state-of-the-art approaches [35, 87] have a significant limitation as will be evidenced in the following sections of this thesis.

Before explaining how the state-of-the-art approaches work, it is crucial to clarify that the above-listed methods index only low-dimensional dataset in order to scale exact kNN queries, but not high-dimensional data. This is because tree-based multi-dimensional index approaches (that will be defined later in detail) that are used in the literature for indexing multi-dimensional dataset are not suitable in high-dimensions. After all, the performance of such indexing methods significantly deteriorates with the dimension of a dataset. A linear increase in dimension has an exponential increase in query response time and memory space requirements; thus in high-dimensional spaces, tree-based multi-dimensional methods give no performance gain compared to that of linear search [47]. However, the recent explosion of mobile devices produces a plethora of low dimensional large-scale datasets; so many researchers show interest in efficiently processing exact kNN in large-scale low dimensional datasets.

Arguably, SHadoop [35] Hadoop MapReduce (MR) based approach is one of the most scalable exact kNN approach. During the query execution process, SHadoop retrieves only relevant data partition from the DFS. At the indexing process, however, the lower size of a partition in SHadoop is determined by the minimum block size of Hadoop Distributed File System (HDFS), in which SHadoop operates. Consequently, in SHadoop, the minimum size of a partition is at least 128MB (default HDFS block size). Regrettably, though, by setting the minimum partition size to such large values has a negative impact on overall query processing time because in spite of only k data points are required to answer a kNN query, at least 128MB -it could be millions of data points- are retrieved from HDFS and load to a memory.

On the other hand, according to the authors, Simba [87] is the best kNN query processing method that operates in the Spark system. Similar to SHadoop, during the query execution process, Simba accesses only relevant data partition to a query. However, Simba defines the lower size of a partition, β, as $\beta = \lambda((1 - \alpha)M/c)$, where λ is a system parameter (usually 0.8) capturing run-time memory overheads, c is the number of cores, M is the total memory reserved for Spark on each worker node, and α is the fraction of M reserved for RDD caching (RDD is a fundamental data structure of Spark). Thus, β is usually in the hundreds of MBs, if not GBs. As the design philosophy of Simba is not meant to optimise disk I/O, during query execution, Simba accesses too many data points, and hence such a design has a negative impact on overall query processing time.

Neither Simba nor can SHadoop set a size of a partition below the default lower size of HDFS block; this constraint is enforced by a design pattern of HDFS and will be discussed

in chapter 3 in detail. At this point, it should be noted that the performance of both state of the art methods is adversely affected by the limitations of the platform in which they operate; thus, when running over HDFS, neither Hadoop MR nor Spark, is suitable for high performance kNN query processing.

Departing from the existing systems that link the size of a partition to the DFS block size, as part of the contribution of the thesis, a coordinator-based kNN query processing method is proposed. The proposed method has two solutions: coordinator with index (COWI) and coordinator with no index (CONI). Both solutions differ from the existing approaches because the way the proposed methods organise, structure, and store a large-scale dataset is entirely different by design. The core foundation of the two solutions rests upon: (i) partitioning a dataset into the smallest possible data partitions that can optimise a kNN query to be efficient, effective and scalable; and (ii) accessing the relevant data partition quickly. To this end, a large-scale dataset is divided into very small-sized chunks (cells) using a well-known multi-dimensional indexing approach, Quad-Tree (QT) [39]; and in both solutions, those cells are stored in No SQL key-value data store HBase table [41].

Dividing a large-scale dataset into small cells produces a very large-sized tree-like data structure that supposed to be served as an index and to be stored in memory. When there is enough space in the coordinator to store the index in memory, the first approach COWI is employed for kNN query processing. But when the size of the index exceeds the available free memory in the coordinator, there are two options: either (i) to increase the size of cells or (ii) to store part of the index in disk. However, as increasing the size of a cell has an adverse effect on query performance, the second approach, CONI, is proposed. CONI stores part of the index in HBase table so that a large-scale dataset can be divided into small cells without any restriction imposed by the amount of free available memory in the coordinator.

Intuitively, accessing an index stored in HBase is more expensive (w.r.t query response time) than that of stored in memory. Accordingly, if enough memory is available in coordinator, COWI has better query response time than CONI, as will be shown later. But, in general, the proposed approaches have achieved up to three orders of magnitude performance gain compared with SHadoop and Simba.

Considering the limitation of COWI, its performance depends on the available free memory in the coordinator, and the limitation of CONI, where access to the index stored in HBase is relatively expensive, it would be better to come up with a new approach that (i) has better or comparable performance comparing to COWI; (ii) does not require to keep space hungry index either in memory or disk; and (iii) is able to partition a large-scale dataset into small-sized cells. To this end, a novel approach, coined Space Transformation Organisation Structure (STOS), is proposed. Departing from the traditional way of indexing a dataset by employing tree-based multi-dimensional indexing methods, a design philosophy of STOS is

grounded upon well-known statistical theories and principles.

STOS by employing statistical learning methods, such as Gaussian Mixture Models, Independence Statistical Copula, and Probability Integral Transformation, an unknown arbitrary distribution of a dataset is transformed to a standard joint uniform distribution. Afterwards, the simplest indexing method, i.e. uniform-grid [16], is used to index the dataset based on the transformed data. STOS has a minute memory footprint that does not grow with the ever-growing data. After all, uniform-grid index method only requires to store in memory a width/height of the equal-sized cells in order to retrieve relevant cells to a query. Moreover, as uniform-grid works incredibly well with uniformly distributed data [73], STOS has the following advantages: (i) quick indexing time, (ii) better space utilisation, (iii) high applicability in higher dimensions (in this context higher dimensions means less than 10-15 dimensions), (iv) minute memory footprint, and (v) better or comparable query response time compared to COWI and up to three order better performance compared to Simba and Shadoop.

In STOS, the dependency of memory space requirement on dimensions is successfully removed; however, query response time depends on dimension. Unable to solve the exponential dependency of query response time on the dimension, researchers assume no efficient solution exists [60] and propose approximated kNN in high dimensional space. But in this thesis, a new method, estimated kNN, is proposed to tackle the dependency of query response time on dimension.

The same statistical parameters to that of STOS are used for space transformation in order to compute estimated kNN. Unlike approximated kNN, the proposed method, without accessing a dataset, tries to estimate values of the actual kNN with high accuracy. Furthermore, the proposed method provides circles (or spheres) centred at each kth estimated result; thus, the actual value of a corresponding k^{th}NN is expected to be found within the circle with some degree of confidence. Similarly, a user can provide a radius (difference in distance between the estimated and actual results that the user willing to tolerate); hence, the proposed method can notify the user of the degree of confidence that the actual results can be located within the user-defined distance from the estimated results.

Similar to the estimated kNN, probabilistic kNN regression is proposed. A vanilla kNN regression predicts the value of a target random variable (RV) by averaging values of a target RV of kNN to a query. However, as stated earlier, in high dimensional data, time elapsed to retrieve kNN increases with dimensions of the data. To this end, probabilistic kNN regression is proposed to predict a value of a target RV by averaging k plausible values of the target RV. The k possible values are probabilistically determined using space transformation technique. Hence the statistical parameters of STOS can be used directly applied in probabilistic kNN. Probabilistic kNN has high prediction accuracy and low query response time.

In contrast, Pythia [8] is a parallel/distributed framework designed for Missing Values (MVs) Imputations. In Pythia, a dataset is distributed across several data nodes. Pythia imputes MVs on the basis of kNN, i.e. kNN is used as a missing value imputation algorithm (MVA) in high dimensional space. The principal idea was that given an input vector with MVs, Pythia *predicts* the most appropriate (relevant) data centres to get involved in the processing of imputing MVs. As Pythia avoids engaging all machines while imputing MVs, only a relevant subset of the entire dataset is accessed. Therefore, in this thesis, the performance of kNN in Pythia, in high dimensional data space, attracts interest. Considering that Pythia does not guarantee to retrieve exact kNN, but experimental results show that the estimation error of Pythia is comparable to that of exact kNN [8].

Each data node in Pythia uses Adaptive Resonance Theory [20](ART), a member of unsupervised machine learning algorithms, to quantise dataset that stores locally. The general process consists the following steps: (1) each data node clusters data, stored locally, using ART and send the cluster heads as a data digest (aka signatures) to a central machine called Pythia. (2) By exploiting the signatures received from all data nodes, Pythia maintains global knowledge regarding the distribution of a dataset across the data-nodes. When an input vector with MVs arrives at the system, using the signature, Pythia predicts the appropriate data-nodes that contains relevant data to the impute vector. Then, Pythia sends the input vector to the selected (appropriate) data-nodes only. Each of the selected data nodes independently execute an MVA - for example, kNN - and estimates values of the MVs; afterwards, Pythia receives estimated values from the selected data nodes and computes the final estimated values by aggregating the estimates received from the data nodes. As Pythia accesses only relevant data partitions while imputing MVs, the kNN algorithm, in high dimensional data space, scales exceptionally well in Pythia.

As Pythia utterly depends on signatures created by ART to identify relevant cohorts (which is very important for the accuracy of a kNN based estimation), in this thesis, the performance of Pythia (w.r.t estimation accuracy and imputation time) is investigated by adopting signatures created by a different clustering algorithm: the Self-Organising Maps (SOM) [51]. Furthermore, the estimation accuracy of kNN in high dimensional space is compared to a more sophisticated MVA: the Expectation-Maximization (EM) [68]. Even though kNN gains slightly better accuracy than EM, in general, both MVA have comparable accuracy.

In contrast, even when accessing only one data-node, Pythia accesses the whole dataset that resides in the data-node. As nowadays a typical disk size is hundredth of Gigabytes if not Terabytes, accessing such huge dataset, even in parallel, might take considerable time and resources particularly when processing large-scale datasets. To alleviate such a problem, more data nodes can be added to Pythia ; but keep adding data nodes with the ever-growing dataset is not economically viable. To this end, instead of accessing the entire dataset that resides in a data-node, a new method that accesses only relevant clusters that reside in appro-

priate data-nodes is proposed in this thesis. The proposed method has a better or comparable estimation error to that of original Pythia. Imputation time of the proposed method is considerable small than that of the original Pythia; moreover, the imputation does not significantly grow with a size of a dataset that resides in a data node or with the number of data nodes in Pythia.

Last but not least, in case of high dimensional data, previous study [17] noted that the relative contrast of the distance of an input vector $\mathbf{i}$ with another vector $\mathbf{c}$ depends heavily on the adopted L_f distance metric where $L_f = \left(\sum_{i=1}^{d} (\mathbf{i}_i - \mathbf{c}_i)^f\right)^{1/f}$; this provides considerable evidence that the meaningfulness of the L_f worsens faster with increasing dimensionality for higher values of f. To this end, in order to study how Pythia (kNN) can be affected by L_f, two distance metrics, the Euclidean distance vs Manhattan distance, are compared.

1.2 Thesis Statement

The rationale of this study is that kNN in very large-scale datasets can and should be computed swiftly. This can be achieved by avoiding irrelevant data access and retrieving only a small and relevant part of a dataset at query time. In order to ensure this, a low dimensional large-scale dataset must be partitioned (indexed) into small chunks or cells using multi-dimensional indexing methods. Furthermore, scarcity of memory can compromise the creation of small partitions as there might not be sufficient memory for storing a large index created by the ever-growing data; thus, using statistical methods, the dependency of an index size on memory can and should be removed. In the case where exact kNN cannot be computed efficiently as a query response time of the multi-dimensional indexing methods increase exponentially with a linear increase in the dimension (the curse of dimensionality), using statistical methods, kNN can be estimated swiftly and with high accuracy as we shall see later.

The main aim of this study consists of:

- studying the effect of accessing small but relevant data partition on the scalability and performance of kNN queries over large-scale data;
- investigating the limitation on indexing, storing and processing of massive dataset of existing methods and identifying the reason why such methods do not access a tiny data partition when executing kNN queries;
- applying statistical learning methods and appropriate big data framework to overcome the limitation of the existing techniques to ensure high scalability and performance of kNN queries that run over large-scale dataset.

In particular, by observing the importance of intra-query and inter-query parallelism, accessing only small and relevant partitions increases the overall system throughput. By drifting away from the existing related works that define the lower size of a partition based on the size of a DFS block, No-SQL datastore (HBase) is used for data storage. HBase allows access to a small data partition. Furthermore, a new indexing method that transforms an arbitrary distribution of a dataset into a joint uniform distribution is proposed. Unlike the traditional tree-based indexing approaches, the proposed method has a very tiny memory footprint and does not significantly grow with the size or dimension of a dataset. The new indexing method along with the HBase datastore allow for partitioning a large-scale dataset into small chunks (cells) and accessing a tiny subset of a massive dataset at query time. Finally, due to the fact that query response time of kNN increases exponentially with dimension, methods that provide estimated results are proposed; the estimated values have high accuracy and can be computed in a few milliseconds irrespective of the number of dimensions of a dataset.

1.3 Summary of Contributions

One of the main issues with distributed/parallel systems is that executing a query usually involves scanning very large amounts of data that can lead to high response times; not enough attention has been devoted to addressing this issue in the context of low dimensional data [7]. The main contribution of the study lies in (i) the practical contribution of improving scalability and performance of kNN query running over parallel/distributed systems, and (ii) the theoretical contribution of showing how statistical learning methods can be adapted to scale kNN query in such setting. In more details, the contributions are:

C1 In state-of-the-art methods, the adverse effect on query performance of accessing a relatively large amount of data is theoretically discussed and practically demonstrated by running extensive experiments.

C2 The design philosophy that underpins kNN query processing over very large datasets is revised; thus, a new way of organising, structuring and indexing multi-dimensional data is proposed. The proposed method, coordinate-based approach, accesses only very small subsets of the datasets; consequently it achieves up to three orders of magnitude lower query processing times than that of state of the art systems.

C3 Especially when opting to access a small subset of a very large-scale dataset, the limitation of traditional tree-based multi-dimensional data indexing approaches is studied comprehensively. Since an index storage space increases with the ever-growing data, the idea of storing and managing part of the index in a No-SQL data store is proposed.

C4 Away from the traditional tree-based multi-dimensional data partitioning methods, a new way of indexing large-scale dataset is proposed on the basis of statistical learning methods. The proposed method ensures: several orders of magnitude lower index-storage space, storage space does not significantly grow with the dataset size and does not grow exponentially with a linear increase in dimension, several orders of magnitude better index building time, easy and fast to recover, and access to a tiny subset of a large-scale data during execution time.

C5 The query response time of all the state-of-the-art approaches, including methods proposed in this thesis, increase with dimension. To this end, a new approach for processing kNN is proposed. The proposed method estimates kNN data points. The estimated kNN are located within a short distance from actual kNN data points and orders of magnitude lower query response time than exact kNN methods, especially in high dimensional data.

C6 As a kNN regression predicts a value of a target RV by averaging values of target RV of kNN to a query, a new probabilistic kNN regression is proposed. The proposed approach has high prediction accuracy and low query response time.

C7 The estimated kNN and probabilistic kNN regression can be used either as stand-alone solutions or in conjunction with an exact kNN processing method, which is proposed in this thesis. Hence, a user can enjoy the flexibility of choosing either estimated results with high accuracy in a short time or exact results with relatively longer time. This can be done without any extra overhead cost of learning additional statistical parameters, that means, only parameters that are needed for computing exact kNN are required for estimating kNN.

C8 kNN as MVA in high dimensional space in Pythia, the framework of missing value imputation, is studied thoroughly. Different space quantisation (some sort of indexing) methods are compared: accuracy of kNN when data is partitioned by the self-organising maps (SOM) is compared to the accuracy of kNN when data is partitioned by adaptive resonance theory (ART). The scalability and performance of kNN under the two space quantisation techniques have no significant difference. Furthermore, the accuracy of kNN as MVA in high dimensional space is studied by comparing to a more sophisticated MVA, the expectation maximisation (EM). As the cost of accessing an entire dataset that resides in a data node of Pythia can be considerably expensive, even in parallel, a new method is proposed. The proposed method, instead of accessing entire data that reside in relevant data nodes, it accesses only few and relevant clusters; this ensures that the scalability of the kNN is achieved without adding data nodes to Pythia.

C9 Last but not least, the effect of distance metrics in the high dimensional space of kNN as MVA is studied in Pythia by comparing the accuracy of kNN algorithms on the basis of the Euclidean vs Mahnabolis distance metrics.

1.3.1 Origins of the Materials

The corresponding publications in peered-review journals and international conferences derived and published from this research are listed below:

- Cahsai, A., Anagnostopoulos, C. , Ntarmos, N. and Triantafillou, P. (2017) *Scaling k-Nearest Neighbors Queries (The Right Way)*. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5-8 June 2017, pp. 1419-1430.
- Cahsai, A. Anagnostopoulos, C., Ntarmos, N., and Triantafillou, P. (2018), *Revisiting Exact kNN Query Processing with Probabilistic Data Space Transformations*, IEEE International Conference on Big Data [submitted Aug 2018]. **The paper was awarded Best Student Paper.**
- A. S. Cahsai, C. Anagnostopoulos (2018) *Scaling-out kNN Regression via Probabilistic and Estimated kNN Query Processing*, 2020-21 IEEE International Conference on Big Data (IEEE BigData 2020-21])[will be submitted Oct 2020-21]
- Cahsai, A., Anagnostopoulos, C. and Triantafillou, P. (2015) *Scalable data quality for big data: the Pythia framework for handling missing values*. Big Data, 3(3), pp. 159–172.

1.3.2 Thesis Structure

The proposed methods are further discussed in the coming chapters as shown in fig. 1.1.

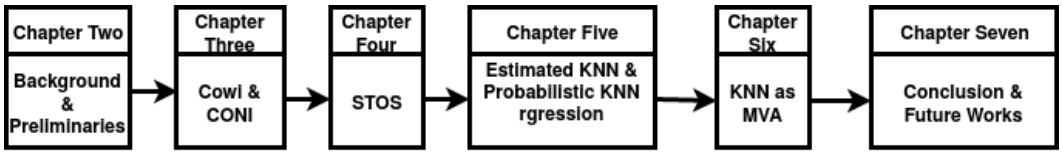


Figure 1.1: Thesis structure

Chapter 2

Background & Preliminaries

2.1 Multi-dimensional Data Indexing Methods

Indexes are used for locating relevant data in a given query without accessing a whole dataset that might have sheer volume. Traditional data indexing methods that have been used for indexing one-dimensional data are obviously not suitable for indexing multidimensional data (m-d for multidimensional), giving rise to several m-d indexing solutions [15, 16, 39, 43, 71, 61].

Those m-d indexing methods can be categorised into four major groups based on how they partition a data space [73]: (1) Uniform decomposition, (2) Non-disjoint decomposition, and (3) Disjoint decomposition.

2.1.1 Uniform decomposition

This [16] method decomposes the data domain space into *equal-sized* partitions (aka cells). Uniform decomposition, also called Uniform grid, divides the domain space into $\prod_{i=1}^{d}(n/\alpha)_i^{1/d} = (n/\alpha)$ non-overlapping cells where d is the number of dimensions, α and n are the average number of data points stored per cell and the total number of data points in the dataset, respectively.

Uniform grid performs exceptionally well when the distribution of the underlying dataset is *uniform*. But when the distribution of the dataset is *non-uniform*, some cells might be empty while others will contain a significantly large number of data points. Uniform decomposition has poor load balance across the cells, and hence has inefficient query performance.

2.1.2 Non-disjoint decomposition

Non-disjoint decomposition methods decompose the domain space into overlapping hyper-rectangles; such methods partition data points based on their clustering relationship and index the partitions hierarchically [73]. Arguably, the most popular non-disjoint decomposition method is R-tree [43].

In R-tree, data points that are located close to each other are represented by a minimum bounding (hyper-) rectangle (coined MBR), that is, the smallest hyper-rectangle that encloses the data points. R-tree is a multidimensional version of the B^+tree; thus R-tree is a balanced tree. Query response time in R-tree might be significantly high due to a high number of overlapping MBRs. Especially, when the distribution of the underlying dataset is skewed, more MBRs overlap and hence query performances is adversely affected. The main cause for increasing the number of overlapping MBRs is that MBRs created from the early-inserted data might not efficiently represent the current data; hence several splitting and re-insertion methods were proposed to address the problem.

The R^* tree [14] improves the insertion mechanisms of R-tree and reduces the overlapping MBRs. It introduces forced reinsertion to redistributed data points more evenly. R^* tree has better query performance than R-tree because it has fewer overlapping MBRs.

2.1.3 Disjoint decomposition

Disjoint decomposition, such as Quad tree[39] and K-d tree[15], divide the domain space into non-overlapping disjoint cells. For example, one of the most commonly used space-driven method is Quad tree. At each split, Quad tree divides a dimension of the domain space into two parts; thus, 2^d disjoint cells are created, where d is the number of dimensions. There are two variants of Quad tree: trie-tree and point-tree. Trie-tree partitions each dimension at mid-point so that the resulting cells have equal size; whereas point-tree divides a dimension of the domain space into two parts, each of which contains equal number of data elements.

2.1.4 kNN Query Processing

A naive approach for searching the exact k closest d-dimensional data points with respect to a given query $\mathbf{q}$ over a dataset of size $n = |\mathcal{D}|$ requires $O(nd)$ time. This also implies $O(ndK)$ time for retrieving the k nearest data points. Nonetheless, one could build a d-dimensional tree over the points of $\mathcal{D}$ to allow to perform efficiently exact kNN search. Such structure is *good* for searches in *low-dimensional spaces*. However, its efficiency decreases as dimensionality grows, and in high-dimensional spaces this structure gives no performance

over naive $O(ndK)$ linear search [47]. Thus, it is worth to note that the tree-based m-d indexing approaches are suitable in low -dimensional space.

Bearing in mind that, most tree-based indexing methods have two steps to compute the exact kNN answer: (Step 1) Computing an initial solution, and (Step 2) Verifying correctness of the said solution. In Step 1, the closest cell (leaf node) to the query point is identified by traversing the index tree. Subsequently, the k-nearest neighbours that reside in that cell are determined. In Step 2, a circle with a specific radius centred at the query point is considered. Any cell that overlaps with the circle is checked to determine if it contains some points, whose distance to the query point is less than any distance of a point that belongs to the initial kNN answer set from Step 1. If such a point exists in any of the candidate cells, then it is inserted into the kNN set and the furthest point is removed.

Many researchers have extensively studied how to compute the optimal radius size. For example [69], also adopted by [7], estimated a radius size through the distance between the query point and the furthest *corner of the cell* which encompass the query point. This is achieved without accessing the dataset but only utilising the information stored in the index. However, this estimated radius size might be quite larger than it should be, and thus a high number of candidate cells may be accessed at query time. A variant to this approach is discussed in [35], where the radius size is estimated by the distance between the query point and the k-th nearest point that lies within the cell that contains the query point. In this case, the dataset must be accessed to compute the radius size. The radius size could be smaller compared to [69] but several data accesses would be required to compute the kNN list.

2.2 Parallel & Distributed Big Data Systems

Hadoop MapReduce: An Overview

Apache Hadoop MapReduce(MR) [31] is a framework for processing large-scale datasets in parallel over several commodity hardware. In MR, a job is a top-level unit of work and might consist of map and reduce phase - the reduce phase is optional.

During the map phase, an input data is partitioned into several partitions using the map task that runs in parallel in a cluster. The map phase emits a key-value paired data that is used as an input to a reducer phase.

The reduce phase processes the key-value (input data to the reduce phase) in parallel and computes the final results. The final results are then stored in a Hadoop distributed file system (HDFS).

HBase: An Overview

Large-scale data can hardly be stored in a centralised server. Even large-scale distributed relational databases are viewed as non-scalable. Thus, typically modern distributed database systems, such as NoSQL databases or DFSs (such as HDFS) are being used. Arguably, HBase [41] is one of the most popular NoSQL databases, offering an implementation of the BigTable[23] model. HBase is highly available and scalable, open source, provides a simple key-value API, and is designed to store large datasets. In this study, HBase has been chosen as the basic data store because it does not need to retrieve the entire DFS block into memory during query execution [79]. A table in HBase is divided horizontally (i.e., at rowkey boundaries) into regions, each of which, in turn, has several HFiles. Additionally, each HFile contains a simple index of the row-keys it contains, and HBase keeps track of which storage nodes and region-servers are responsible for every region. These features enable HBase to support efficient random data access.

Spark: An Overview

Spark [92] is another cluster-based batch-oriented big data-parallel processing platform. The main advantage of Spark is the ability to run computations in memory. Spark defines resilient distributed datasets (RDD) representing a collection of items distributed across many nodes that can be manipulated in parallel. Spark has several extensions that provide different features: Spark SQL [10] for working with structured data, Spark Streaming [93] for processing of live streams of data, MLlib [59] with machine learning algorithms, and GraphX [88] for manipulating graphs.

Data partitioning Methods in Parallel Distributed Systems

All the state-of-the-art parallel/distributed methods process large-scale data in parallel across a cluster and can effectively solve several complex queries. However, when processing some kinds of queries, such as kNN, accessing the whole dataset in parallel to retrieve only k data points wastes time and precious resources. Hence, to avoid accessing the entire dataset during query processing, m-d index methods are built over a dataset stored in a DFS (distributed file system).

However as most of the m-d index approaches were originally designed to run in a centralised system, to adapt them in the parallel/distributed approaches, several data partitioning methods have been proposed in the literature. The most popular data partitioning methods are summarised below as discussed in [73].

Uniform Data partitioning: Uniform data partitioning method partitions an input data into non-overlapping equal sized hyper-rectangles (cells) where the user decides the number and size of the cells. The index building time of uniform data partitioning methods is minimal due to the simple process and computations involved [73]. However, uniform data partitioning, when indexing non-uniformly distributed data, has bad load balancing as some nodes complete their task early and sit idle, waiting for other heavily loaded nodes to finish their work.

cluster-based data partitioning: Arguably, the most popular cluster-based data partitioning method is [21]. This method adopts tries to identify the number of clusters in a dataset using the k-means iterative clustering algorithm and Bayesian-Information Criteria (BIC). Afterwards, the dataset is clusted using the Gaussian Mixture Models. However, unlike the methods proposed in this study that scans a whole dataset only once to partition the data, [21] do multiple scans over the whole dataset using several MapReduce jobs to iteratively cluster the dataset. As the processing cost of such multiple iterations is extremely high, indexing time of this method is high compared to the other indexing methods.

Random-Sampling based data partitioning: Random-Sampling based data partitioning methods initially build an in-memory R-tree based on a small data sample drawn uniformly at random from the input data. Most of the time the default size of the sample is 1% of the input data with a maximum capacity of 100MB to ensure it fits in memory. Afterwards, the sample data bulk-loaded to the in-memory R-tree using a Sort-Tile-Recursive (STR) packaging method [54]. Finally, the whole dataset is partitioned based on the boundaries of the leaf-nodes of the in-memory R-tree. However, according to a recent survey [73], such data partitioning methods work fine for uniformly distributed data, but not for non-uniformly distributed data; furthermore, the same observations were noted when running some experiments in this thesis, as we shall see later.

Quad-tree based data partitioning: Partitioning a dataset in parallel/distributed approaches using QT starts by creating in-memory QT, whose maximum capacity per node is equal to s/p where s is the sample size, and p is a total number of partitions in which a large-scale data has to be divided. Afterwards, the large dataset is partitioned based on the in-memory QT. If any of the resulting leaf nodes of the QT contains more than a user-defined threshold, then the leaf node has to keep splitting until all the resulting leaf-nodes capture the most user-defined threshold. Due to the non-overlapping nature of the QT cells, this method works well even with non-uniformly distributed data.

According to [73], the QT provides better data proximity, efficiency for search queries, and low network transfer overhead as compared to other data partitioning methods. However, QT requires high index storage space, index building time, and has poor applicability in high dimensions (up to 10-15 dimensions).

Chapter 3

Scaling exact kNN Queries: with & without in-Memory Index Tree

3.1 Introduction

Nowadays, modern devices such as smartphones, space telescopes, medical devices, and moving objects are generating a plethora of low-dimensional data; the data of interest either have low dimensionality or can be indexed using few dimensions. For example, 2-3 dimensional X-ray images are produced by medical devices at the rate of 50 PB per year [1]; daily around 10 million geotagged tweets are generated on Twitter [2]; and the NASA archive of satellite earth images is more than 500 TB in size and increases daily by 25 GB [62]. An efficient processing of the ever-growing data provides timely and valuable knowledge across different scientific disciplines; for example, epidemiologists use spatial analysis techniques to identify cancer clusters, track infectious diseases, and determine drug addiction, while meteorologists simulate climate data through spatial analyses, and news reporters detect events on the basis of geotagged tweets.

Unfortunately, the traditional off-the-shelf data processing tools, which are typically designed to process much smaller datasets stored in a centralised system, are unable to handle the fast-growing data. To fill this gap, several parallel/distributed data-processing frameworks have been proposed for *big data* analytics.

Arguably, the two most popular frameworks for big data analytics are Hadoop-MapReduce (MR) [31] and Apache Spark [92]. MR and Spark use a cluster of commodity hardware for the parallel processing of big data analytics. Unfortunately, such frameworks are not a panacea for all ad hoc big data query processing, yet many researchers continue to propose MR- and Spark-based solutions even when they are flawed. For example, assume that one executes a k nearest neighbour (kNN) query over large-scale data by using either vanilla MR

or Spark SQL [10]. As the value of k is small (most of the time) and such frameworks access the entire dataset in parallel to retrieve only k data elements, accessing a massive part of a large-scale dataset that makes no contribution to the final answer to the kNN query wastes time and resources. Thus, recently, many researchers have become interested in finding an efficient method for processing the exact kNN queries over low-dimensional large-scale datasets.

Arguably, MR-based SHadoop [35] and the Spark-based Simba[87] are among the recent most popular contenders for processing kNN over low-dimensional large-scale datasets. The core design philosophy of both of these approaches are as follows:

- divide a dataset into several partitions (subsets), each of which contains data elements that are located relatively close to each other in the Euclidean space;
- build a global multi-dimensional index over all the partitions to prune out irrelevant partitions at the query time; and
- build a local multi-dimensional index within each partition to avoid a linear scan of data elements that reside in a selected partition when processing a query.

As both the approaches prune out irrelevant partitions and avoid a linear scan of the data elements that reside in a relevant partition, they have a lower query processing cost than vanilla MR or Spark SQL.

However, a lower size of a partition in SHadoop and Simba is dictated by the settings in which a particular method operates. For example, as SHadoop operates within the Hadoop system, the minimum size of a partition is set to the block size of the Hadoop distributed file system (HDFS). The default size of the HDFS block is 128 MB. Similarly, a lower size of a partition, denoted by β, in Simba is computed as follows: $\beta = \lambda((1 - \alpha)M/c)$, where λ is a system parameter (usually 0.8) that captures run-time memory overheads, c is the number of cores, M is the total memory reserved for Spark on each worker node, and α is the fraction of M reserved for RDD caching. Thus, β is usually in the hundreds of megabytes if not gigabytes. Regrettably, limiting a lower size of a partition to such a large value has an adverse impact on the kNN query processing, as illustrated in the following example.

Motivating Example

Consider that a kNN query is executed over a dataset of spatial points stored in HDFS. A point is represented in a 2-d space by a pair of numbers: a number on the x-coordinate and a number on the y-coordinate. Here, the size of a point is 16 bytes as each coordinate has a double-precision floating point number, thus needing $2 \times 8 = 16$ bytes in total. During the

kNN query execution, when $k = 10$ (resp. $k = 100$ or $k = 1000$), the optimal would be that only 160 (resp. 1600 or 16000) bytes of data is to be retrieved as a final answer to the query. Unfortunately, at least one partition (128 MB), containing $\approx 8.4 \times 10^6$ points must be read from disk. This demonstrates that having such large partitions is highly inefficient because of the following: (i) despite the fact that most of the time, the value of k is small, a high volume of data is read from disk, transferred over the network, loaded into memory and processed; and (ii) data elements located relatively *far* from each other in the Euclidean space might be stored together as members of a partition; as the lower size of a partition increases, the chance of compacting *non-neighbours* data elements into one partition increases. Consequently, during query processing, it is inevitable for data elements that do not contribute to the final kNN answer to be read from disk, transferred over the network, and loaded into the memory.

The rationale of SHaddop and Simba to trade the performance for maintaining a large partition is guided by [5, 48, 95, 94]. These guidelines dictate that creating too many small files (one per a partition) can quickly overload the central node (aka NameNode) of a cluster. NameNode stores and manages the meta-data of all the partitions in HDFS; threfore, overloading the NameNode with the meta-data of too many small partitions compromises the overall health of a cluster. Accordingly, a lower size of a block in HDFS (in this case partition) should be at least 128MB.

Contrary to the existing systems, in this part of this thesis, a *coordinator-based approach* is proposed. The proposed method partitions large-scale data into several small chunks (hereinafter referred to as *cells*). The small size of the *cells* is typically designed to reduce accessing data elements that do not contribute to the final answer to a kNN query; thus, the proposed method has a considerably lower overhead cost than SHadoop and Simba, as will be shown later in the chapter. The cells are stored in the HBase key-value data store table. HBase allows storing and accessing of cells without compromising the overall health of a cluster. A Quad-tree (QT) [39] is used to index a dataset, but it is worth to note that any multi-dimension index approach can be used instead. QT is an arbitrary choice.

In contrast, tree-based indexing methods are widely used,[35, 5, 87, 7], for partitioning low-dimensional data and building a multi-dimensional index over a dataset. Although tree-based indexes have often been referred to as multi-dimensional indexes in the literature, their performance significantly deteriorates with an increase in the dimensionality of the dataset. A linear increase in dimension results in an exponential increase in the query response time and the memory space requirements [47]. For example, for a dataset that has 10 to 20 dimensional data (depending on the total number of data elements) the entire dataset is accessed when a tree-based multi-dimensional index is used; the index can not avoid a linear scan of the dataset. For more details, refer to [47].

Unable to solve the exponential dependency on the dimension, researchers have assumed

that no efficient solution exists [60] and proposed approximated kNN in a high-dimensional space. However, approximated kNN is not in the scope of this chapter. Thus, because of the exponential dependency of tree-based indexing approaches on the dimension, this part of the thesis, such as [5, 35, 63, 44, 91, 7, 87], focuses only on processing the exact kNN over large-scale low-dimensional data. Note that to avoid confusion regarding the name, in this thesis, in alignment with the literature, tree-based index methods are referred to as multi-dimensional or multivariate indexing approaches; however, please bear in mind that such methods are appropriate in a low-dimensional space only.

3.2 Contribution

The contributions of the proposed method are as follows:

- Revisit the design philosophy that underpins the exact kNN query processing over very large datasets, going against the grain and the state-of-the-art methods.
- Offer a different method to index and organise the dataset that enables accesses to only very small subsets of the dataset.
- Offer coordinator-based query processing algorithms that exploit the above and which on the whole ensure the following:
  - High performance – up to three orders of magnitude lower query processing times than the state-of-the-art methods.
  - High scalability – ensuring that compute-storage-network resources are utilised efficiently, for datasets of various sizes, also ensuring high scalability.
- Offer a strategy to handle updates, in case the datasets grow/shrink dynamically.
- Offer an extensive performance evaluation of the proposed approach versus the state-of-the-art methods (Spatial Hadoop and Simba), which substantiates and quantifies the performance and scalability claims of the proposed approach.

This part of the research is published in the International Conference on Distributed Computing Systems:

- Cahsai, A., Anagnostopoulos, C., Ntarmos, N. and Triantafillou, P. (2017) *Scaling k-Nearest Neighbors Queries (The Right Way)*. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5-8 June 2017, pp. 1419-1430.

3.3 Related Work

Different methods have proposed for efficiently processing kNN queries over large-scale datasets by using parallel/distributed systems. This section summarises the related works.

Hadoop/MR-Based Approaches

Hadoop GIS [5] is a scalable and high-performance spatial data warehousing system for running large-scale spatial queries in Hadoop. However, Hadoop GIS only supports two-dimensional data. The state-of-the-art SHadoop [35] divides a dataset into equal-sized partitions: the minimum size of a partition is the same as the smallest recommended size of an HDFS block. SHadoop uses two indexes: a global index and a local index. During query processing, the global index is used to prune out the irrelevant data partitions, whereas the local index is exploited to avoid the linear scanning of data elements that reside in a relevant partition. In order to answer kNN queries, SHadoop might require two MR jobs to ensure the correctness of the final kNN answer. Interestingly, even though the academic papers that describe SHadoop do not discuss it, the source code that the authors have made available includes a non-MR approach for computing kNN. This implies that they also realise the tension between a distributed operation and high performance: initialising an MR job has a high overhead cost. However, it should be clarified that SHadoop as a whole is a good step forward for scalable spatial queries, offering an overall system for many types of spatial queries and not just kNN queries. The point made in this work is that the SHadoop approach is lacking in terms of performance for kNN query processing and that the proposed approach reconciles the performance-scalability tension better. Nonetheless, to be fair, the performance of the proposed method will be compared to that of both variants of SHadoop (MR and non-MR). The proposed method does not discuss other types of spatial queries, such as kNN joins and spatial joins.

AQWA[7] is another recent method for KNN query processing. AQWA splits the dataset into many partitions. Like SHadoop, AQWA sets the minimum size of a partition to a minimum block size of the underlying DFS. Unlike SHadoop, AQWA only has a global index and does not build a local index within partitions. Thus, a linear scan of all the data elements that reside in a relevant partition is required when computing KNN. Therefore, it has a significantly extra CPU time overhead as compared to SHadoop.

Approaches on HBase

Several HBase-based methods have been proposed for efficiently computing the exact kNN queries over large-scale low-dimensional datasets. The MD-HBase system [63] builds in-

dexes over a dataset stored in HBase by using k-d trees and Quad-Trees (QTs) and uses Z-ordering to convert multi-dimensional keys into one-dimensional row-keys. Similarly, [45] proposed a novel key-formulation schema using an R+ tree over a dataset stored in HBase. The main focus of these works was on the design of efficient HBase row-keys, and they did not pay much attention to access the smallest relevant data elements as much as possible while processing the kNN queries.

HGrid [44] builds a multi-dimensional composite index over a dataset using QT and a regular grid and stores the indexed data in an HBase table. HGrid adopts QT to partition the domain space of a dataset into several sub-spaces. Each sub-space has a unique corresponding leaf-node in the QT. Each leaf-node (sub-space), in turn, is further divided into many cells by using a regular grid. At the end of the indexing process, each leaf-node points to a unique row in the HBase table and the columns of a row contain the cells of the corresponding leaf-node. HGrid stores a small number of data elements in a column in order to reduce irrelevant data access during query processing. However, the approach proposed in this work differs from HGrid in several ways such as the follows: (1) HGrid does not include a systematic method of ensuring that a minimal number of data points is accessed. (2) HGrid tries to keep the size of the QT small in order to save memory that is required to store the QT tree. However, HGrid adopts a regular grid to partition each leaf-node; therefore, many columns are created in a corresponding row. Unfortunately, as the number of columns increases (e.g. above several hundred), the query performance deteriorates significantly [44].

Spark-Based Approaches

Spark [92] is another parallel/distributed batch-oriented big data processing platform. The main advantage of Spark is the ability to run computations in memory. Spark defines resilient distributed datasets (RDD). RDDs represent a collection of items distributed across many nodes that can be manipulated in parallel. Spark has several extensions that provide different features: Spark SQL [10] for working with structured data, Spark Streaming [93] for the processing of live streams of data, MLlib [59] with machine learning algorithms, and GraphX [88] for manipulating graphs.

In the literature, several Spark-based kNN query processing methods have been proposed: GeoSpark [91], SpatialSpark [90] (kNN joins and spatial joins over geometric objects), and Simba [87]. Simba [87] extends Spark SQL by adding tree-based multi-dimension indexes and exploits the indexes for efficient query planning and execution. Simba builds a global and local index to reduce irrelevant data access during query processing. According to its authors, Simba has the best performance among the Spark-based solutions mentioned above. Because of this fact, in this thesis, the proposed solutions are compared to Simba.

In contrast with the proposed approach, Simba has two drawbacks. First, the main focus

of Simba is to reduce the CPU cost [87], and it sidelines reducing the disk and network IOs. Thus, during query processing, Simba loads into the memory large RDDs that contain a considerably large number of data elements. Second, when pruning out the irrelevant partitions, Simba generously estimates the relevant subspace and loads into the memory all RDDs located within the estimated subspace as the candidate RDDs. Simba attempts to estimate the relevant subspace by drawing a circle centred at the query (q), but as the radius of the circle is computed on the basis of the distance between q and the furthest corner of the closet RDD, the circle covers a relatively large area that contains a considerably large number of candidate RDDs. Accordingly, Simba processes a relatively high number of data elements; this issue will be discussed in more detail later in the chapter.

3.4 Definitions

Definition 1. *A cell C in a d-dimensional space $\mathbb{R}^d$ is defined by the following triplet:*

$$C := \langle \mathbf{w}, r, |C| \rangle,$$

where $\mathbf{w} = [w_1, \ldots, w_d] \in \mathbb{R}^d$ is the lower boundary point, $r > 0$ is a fixed width in each dimension, and $|C|$ refers to the number of d-dimensional points in the cell.

Definition 2. *A grid G in a d-dimensional space $\mathbb{R}^d$ is a set of m non-overlapping cells $G = \bigcup_{i=1}^{m} \mathcal{C}_i$.*

Definition 3. *A query point $\mathbf{q} \in \mathbb{R}^d$ is a d-dimensional vector: $\mathbf{q} = [q_1, \ldots, q_d]$; a point $\mathbf{p}$ in grid G is a d-dimensional row: $\mathbf{p} = [p_1, \ldots, p_d]$. The Euclidean distance between query $\mathbf{q}$ and point $\mathbf{p}$ is as follows:*

$$\|\mathbf{q} - \mathbf{p}\| = \left(\sum_{i=1}^{d} (q_i - p_i)^2 \right)^{\frac{1}{2}}.$$

Definition 4 ([69])**.** *The minimum distance of a query point $\mathbf{q}$ from a given cell $C \in G$ with a lower boundary point $\mathbf{w}$ and width r, denoted by $f(\mathbf{q}, C)$, is as follows:*

$$f(\mathbf{q}, C) = \|\mathbf{q} - \mathbf{s}\|,$$

where $\mathbf{s} = [s_1, \ldots, s_d]$ and

$$s_i = \begin{cases} w_i, & \text{if } q_i < w_i; \\ w_i + r, & \text{if } q_i \geq w_i + r; \\ q_i, & \text{otherwise.} \end{cases}$$

Definition 5. *Given $m > 0$, a dataset $\mathcal{D} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{|\mathcal{D}|}\}$ of d-dimensional points is divided into m partitions $\mathcal{D}_i, i = 1, \ldots, m$, such that it holds the following: $(\mathcal{D} = \bigcup_{i=1}^{m} \mathcal{D}_i) \wedge (\mathcal{D}_i \neq \emptyset) \wedge (i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset)$.*

$|\mathcal{D}|$ is the cardinality of the set $\mathcal{D}$.

Definition 6. *Given $\alpha > 0$, the upper-bound of the points stored in a partition $\mathcal{D}_i$ is $\alpha \geq |\mathcal{D}|/m$, where m is the number of partitions.*

Definition 7. *Given a partition $\mathcal{D}_i$, cell C_i is the smallest (sub)space within which all the points of $\mathcal{D}_i$ lie.*

Definition 8. *Given a balanced tree data structure, let x denote the maximum number of children per node. In a tree of height h, the total number of nodes z and the number of leaf nodes l are, respectively, as follows:*

$$z = \sum_{i=0}^{h} x^i \text{ , } l = x^h.$$

Definition 9. *Given a query point $\mathbf{q}$ and a dataset $\mathcal{D}$, the k nearest neighbours (kNN) of $\mathbf{q}$ is the set $\mathcal{A}$:*

$$(\mathcal{A} \subseteq \mathcal{D}) \wedge (|\mathcal{A}| = k) \wedge (\forall \mathbf{p} \in \mathcal{A}, \forall \mathbf{p}' \in \mathcal{D} \setminus \mathcal{A}, \|\mathbf{p} - \mathbf{q}\| \leq \|\mathbf{p}' - \mathbf{q}\|).$$

Definition 10. *If the total number of data points in a uniformly distributed domain space is $|\mathcal{D}|$, and α data points are needed to be stored per cell, the domain space can be partitioned into $|\mathcal{C}| = |\mathcal{D}|/\alpha$ equal-width cells. The cell width r is computed as $r = w/(|\mathcal{C}|)^{1/d}$, where w is the width of the uniform domain space and d is the number of dimensions.*

Definition 11. *Let a vertical or horizontal closed interval on a number line in a one-dimensional space start at 0 and be divided into n finite consecutive half-open smaller intervals, each of which has equal-length r. For a given random number q that lies on the ith small interval, the starting number of the ith interval is defined by $\lfloor \frac{q}{r} \rfloor \cdot r$.*

3.5 Rationale

The primary focus of this part of the thesis is on improving the overall scalability of the kNN query by (i) designing small cells that contain few data elements and (ii) retrieving few but necessary cells when processing a query. Next, the rationale of determining a small cell size and of identifying the relevant cell is presented.

3.5.1 Cell Size Determination

When indexing a dataset, if a leaf-node of a quadtree (QT) [39] contains more data points than the user-defined threshold α, the leaf-node divides into 2^d child nodes, where d is the number of dimensions of a dataset. QT partitions a dataset $\mathcal{D}$ into m leaf-nodes, each of which contains at most α data points.

At the end of the data partitioning process of the proposed approach, data elements that belong to a leaf-node are stored in the corresponding row in the key-value data store HBase table. A leaf-node maintains only a pointer to the corresponding HBase row; no data element is stored in the QT. In the coordinator, only the tree-like data structure of the QT is stored in the memory to serve as the index. Each HBase row contains a small partition, $\mathcal{D}_i$, of the dataset $\mathcal{D}$ such that $\mathcal{D}_i \subset \mathcal{D}$ and $|\mathcal{D}_i| \leq \alpha$; eq. (3.1) shows the upper bound on the cell size α, given that $|\mathcal{D}|$ is the total number of data elements in the dataset and x^h is the total number of leaf-nodes (or rows):

$$\alpha \geq |\mathcal{D}|/x^h, \tag{3.1}$$

where x is the maximum number of children per node and h is the height of a QT.

As explained earlier, the maximum number of points, α, stored in a cell (row) has a significant impact on the query response time; thus, the value of α has to be reasonably small. The higher the value of α is, the higher is the query response time as there are more points to consider. In contrast, a small value of α improves the query response time but forces the leaf-nodes to split more frequently and increases the size of the QT, as a result more memory space is needed in the coordinator to accommodate the index. For large-scale datasets, this poses a significant scalability problem at the coordinator.

Question 1: How small can the value of α be? The value of α is dependent on the amount of available memory at the coordinator, β. As the value of α must be known before the data partitioning process (constructing the QT), equation 3.1 should be defined in terms of β. Hence, α should be defined as a function of β, i.e. $\alpha = \alpha(\beta)$. Thus, as explained in Lemma 1, equation 3.5 defines α in terms of β.

Lemma 1. *Let β, b, x, and z be the total available memory, the size of a node of the tree in bytes, the maximum number of children per node, and the total number of nodes in a tree, respectively. The upper bound on the number of points α that can be stored in a cell (leaf-node) is as follows:*

$$\alpha(\beta) \geq |\mathcal{D}| \times x^{1-log_x((\beta/b)(x-1)+1)}$$

Proof. The total number of nodes in a given tree is as follows:

$$z = \sum_{i=0}^{h} x^i = 1 + x + x^2 + ... + x^h \tag{3.2}$$

Moreover, the value of z can be based on the maximum available free memory, β, and the size of a node of a tree in bytes, b, i.e.

$$z \leq \frac{\beta}{b} \tag{3.3}$$

From (3.2) can obtain that:

$$\begin{aligned} z &= \frac{(x^{(h+1)} - 1)}{x - 1} \iff \\ (h+1)log_x x &= log_x(z(x-1)+1) \iff \\ h &= log_x(z(x-1)+1) - 1 \end{aligned} \tag{3.4}$$

Given that $\alpha(\beta) \geq |\mathcal{D}|/x^h$, substituting z from (3.3) yields the following:

$$\alpha(\beta) \geq |\mathcal{D}| \times x^{1-log_x((\beta/b)(x-1)+1)} \tag{3.5}$$

□

Equation 3.5 shows that the value of β increases when the value of α decreases. Therefore, the value of α decays exponentially w.r.t β. Fig. 3.1 shows that $\alpha(\beta)$ is an exponential decay function ($|\mathcal{D}| = 90 \cdot 10^9$, $x = 2$, $b = 32$ bytes).

3.5.2 Candidate Cells Determination

Computing the exact kNN requires accessing more than one cell when either the closest cell to a query contains less than k data elements or some data points that should be part of the kNN answer reside in the neighbouring cells of the closest cell.

Question 2: Is it possible to identify the *relevant* cells that contain the final kNN answer without accessing the entire dataset?

If so, w.r.t improving the query response time:

Question 3: How can the smallest possible number of cells be identified?

Arguably, in the literature, two popular methods, which are adopted by [35] and by [7], are widely used to identify all the relevant partitions to a query. Both the methods identify the closest leaf-node to a query by traversing the index tree stored in the memory. Each

leaf-node contains the following: (i) a pointer, (ii) a counter, and (iii) a width. The pointer has information on how to access the corresponding partition; the counter indicates the total number of data elements stored in the corresponding partition. When the closest partition has less than k data elements, using the width of the closest leaf-node, we can select the second closest cell, the third closest cell, and so on, until at least k data elements exist between the selected closest cells [69]; see definition 4.

After a minimum number of closest cells that contain at least k data elements are selected, a circle centred at the query with radius ρ is drawn. Then, all the leaf-nodes that overlap the circle are selected as the candidate cells; by accessing only the candidate partitions (that correspond to the selected leaf-nodes), we can guarantee the exact kNN answer to a query located at the centre of the circle can be computed.

The two most popular methods that are used to identify the relevant partition compute ρ differently. A circle created by one of the methods is not the same as a circle created by the other method, even when processing the same query. For more explanation, without any loss of generality, assume that the closest cell to a query has more than k data elements. The method adopted by [35] computes ρ in three steps. Step (i) Data elements that reside in the closest cell are retrieved from a DFS. Step (ii) From the data elements retrieved in step (i), the local k^{th} nearest neighbour to the query is identified. Step (iii) The value of ρ is set to the distance between the query and the local k^{th} nearest neighbour, which is identified in Step (ii). However, the other method, adopted by [7], computes ρ on the basis of the distance between the query and the furthest corner of the closest cell. As the closest cell is the closest leaf-node in the index tree, the latter approach avoids accessing data from DFS when computing ρ. However, as the distance between a query and the furthest corner of the closet leaf-node is longer than the distance between a query and any data element that resides within the leaf-node, the latter approach produces a larger circle than the first approach.

Even though the earlier method has tighter ρ, it produces a sufficiently big circle to compute the exact kNN, not a single data element that is assumed to be a member of the kNN answer is located outside this circle. For example, for the sake of simplicity, consider that the closest cell contains more than k data elements. When ρ is computed on the basis of the distance between a query and the kth nearest neighbour that resides in the closest cell, already k locally closest data elements are retrieved from the closest cell. However, other data elements that reside in the adjacent cells might be located at a closer distance to the query than the k data elements selected from the closest cell. In order to identify whether such data elements exist, accessing data elements that overlap the circle is sufficient. As the distance between any data element that resides outside the circle and the query, located at the centre of the circle, is always longer than the radius ρ, we do not need to check for data elements that reside outside the circle as there are at least k data elements within the circle.

However, as initialising a MapReduce job has a high overhead cost, the method that has a bigger ρ has a comparative advantage in a MapReduce environment; because as it uses estimated radius, only one MapReduce job is needed to compute the final answer. However, computing kNN with the method that has a tighter ρ requires two MapReduce jobs: one MapReduce job to retrieve data to compute ρ and the other MapReduce job to compute the final answer. Thus, the latter approach has a higher overhead cost and query response time in a MapReduce environment.

In this work, however, the method with a tighter ρ was adopted because of two crucial facts. (F1) No expensive process, such as initialising a MapReduce job, is involved when accessing data from HBase. As HBase has a minimal overhead cost of accessing data, accessing an HBase table twice per query has an insignificant overhead cost. (F2) Recall that the primary focus of this work is to reduce the accessing of irrelevant data. Hence, a method that computes the tightest possible value of ρ resonates with the purpose of this work. A smaller ρ creates a smaller circle that overlaps with fewer cells that in turn contain fewer data elements to be processed. However, as the other method estimates ρ generously, it ends up accessing more irrelevant data.

To further analyse the effect of ρ on the query response time and on the average number of data points accessed per query, the experimental results are presented in Figure 3.2 and Figure 3.3, respectively. Figure 3.2 shows that there is a strong positive correlation between ρ and the query response time; i.e. as the value of ρ increases, the query response time also increases. As illustrated in Figure 3.3, there is also a positive correlation between ρ and the number of points that are accessed during the query time. Hence, in order to have a smaller ρ, the proposed approach calculates the exact value of ρ by accessing all the points that reside in the closest cell. The results shown in both the figures were computed over a synthetic dataset, that contains 7×10^7 data points, generated on the basis of a bivariate normal distribution with mean $[100, 100]^T$ and covariance $\begin{pmatrix} 50 & 0.9 \\ 0.9 & 50 \end{pmatrix}$;

3.6 First Solution: Coordinator With In-Memory Index Tree

3.6.1 Overview

Armed with knowledge, explained in the previous sections, in this work, a coordinator-based distributed kNN query processing approach is proposed. The proposed approach, coined Coordinator With Index (COWI), uses a QT to partition a large-scale dataset and HBase to store data.

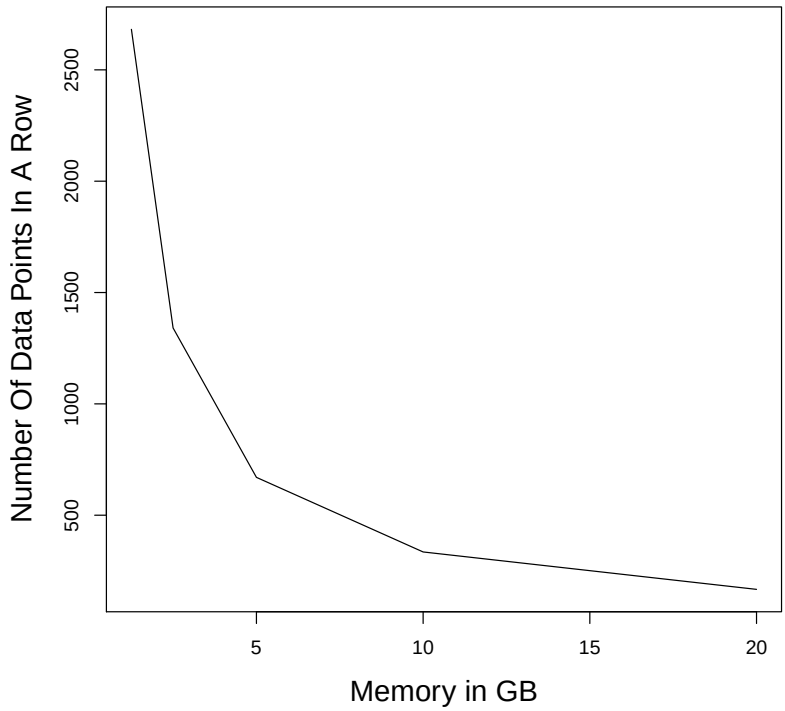


Figure 3.1: As β increases, α decreases exponentially.

Recall that QT is widely used in the literature to partition low-dimensional data. QT iteratively divides a domain space into several sub-spaces. Thus, a geometrical structure of the leaf-nodes of a QT can be represented as a grid, as shown in Figure 3.4. A cell of a grid, $\mathcal{C}_i \subset G$ (G represents a domain space), corresponds to the ith leaf-node of a QT; a one-to-one mapping exists between the cells of G and the leaf-nodes of the QT. Thus, a cell $\mathcal{C}_i$ contains less than α data elements; i.e. $|\mathcal{C}_i| < \alpha$, and hence, $\mathcal{G} = \bigcup_{i=1}^{p} \mathcal{C}_i$.

Moreover, recall that as COWI is designed to process large-scale data, accommodating an entire dataset in the QT, which resides in memory, is infeasible. Thus, as depicted in Figure 3.4, data points that belong to cell $\mathcal{C}i$ are stored in the corresponding row in the key-value HBase table. Only the tree-like data structure of the QT is stored in the main memory of the coordinator to serve as an index. In the memory, every leaf-node of the QT stores (i) a pointer: a row-key to a corresponding row in HBase table; (ii) a counter: the total number of data elements residing in the corresponding row, and (iii) a width: when less than k data elements are stored in the closest cell, the width is exploited to identify the closest neighbours.

QT can efficiently index a dataset generated by an arbitrary distribution [11]. During the data partitioning process, QT divides the highly populated regions more rigorously than the less populated regions. Partitioning a dataset that has a skewed distribution creates an unbalanced

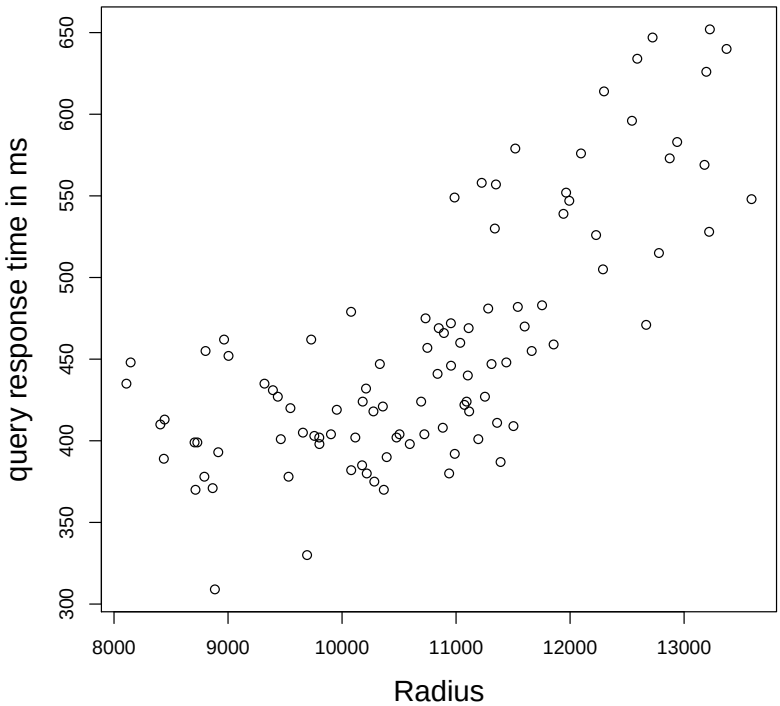


Figure 3.2: Query response time (in milliseconds) vs. radius ρ.

QT. As shown in Figure 3.4, cells that represent the geometrical structure of the leaf-nodes of the unbalanced QT have an unequal size but contain the same number of data elements on average. In contrast, a balanced QT is produced after partitioning a dataset that has a uniform distribution. Intuitively, all the cells that represent the leaf-nodes of a balanced QT are of equal size and contain the same number of data elements on average.

As a QT can partition both types of datasets into several cells that contain less than α data points, the average query processing time of both the datasets is expected to be relatively the same as long as, on average, the same number of cells (rows) are accessed per query. Recall that a circle centred at a query with radius ρ is used to identify cells that must be accessed to answer a kNN query correctly. Thus, particularly for the dataset generated by a skewed distribution, because of the unequal size of cells, determining the value of ρ by the distance between a query and the furthest corner of the closest cell might produce a relatively large circle (because of the large ρ), and more irrelevant cells that have an adverse effect on the query response time might be accessed. Intuitively, identifying the relevant cells by using the other method that has a tighter ρ (smaller circle)see the previous sectionprunes out the irrelevant cells much better. This is another reason why a method with a tighter ρ was adopted in this work.

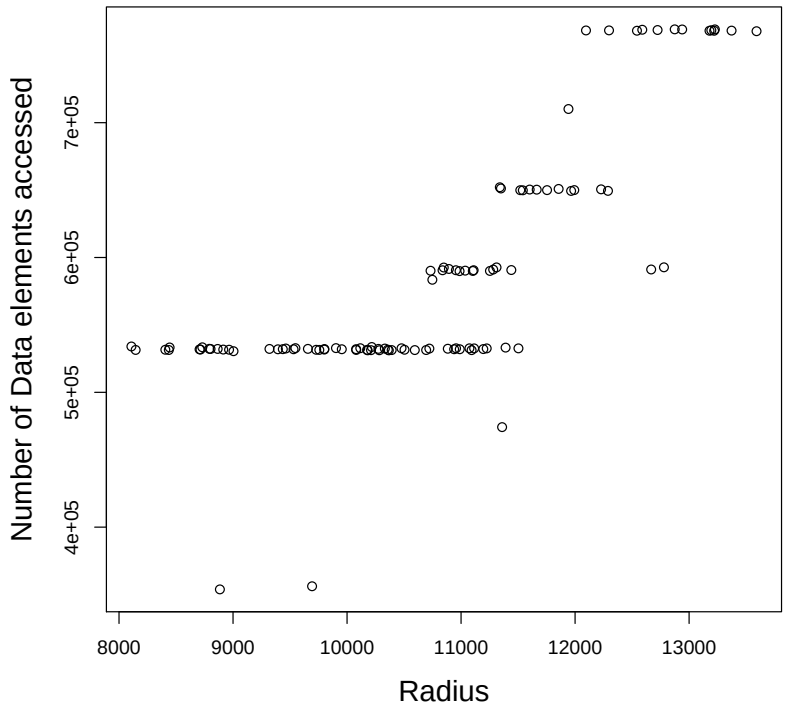


Figure 3.3: Correlation radius (ρ) with (a) query response time and (b) number of accessed data points.

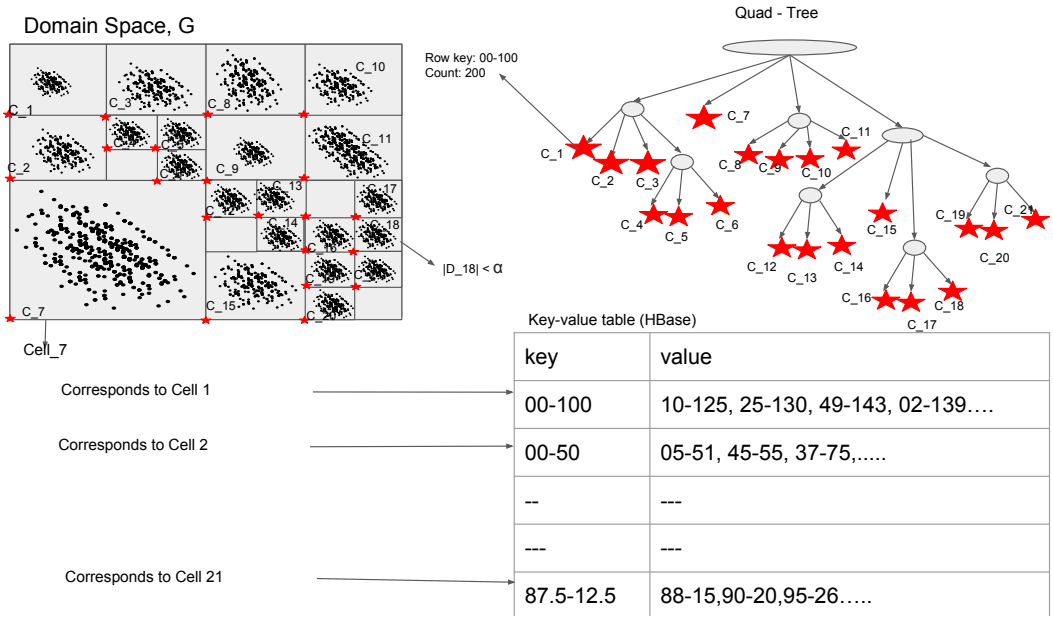


Figure 3.4: Overview of the system for kNN query processing.

3.6.2 Quad-Tree Index Construction

Traditionally, QT was designed to partition a small dataset that resides in the memory of a centralised system. However, constructing QT over a large-scale dataset is a time-consuming

and complex process. Thus, COWI adopts MapReduce to construct QT over a large-scale dataset stored in parallel/distributed systems.

Constructing a QT in MapReduce involves the following four steps: (1) draw sample data from a dataset, as in [34]; (2) based on the sample data, construct in-memory *base QT*; (3) based on the in-memory *base QT*, build a local QT over each mapper that executes a part of the entire dataset; and (4) combine the local QTs to construct the final global QT. In the next few paragraphs, these steps are explained in detail.

Step-1 In this step, a MapReduce job is executed to perform two actions: (i) to draw a uniform random sample with sampling ratio δ and (ii) to count the total number of data elements in the dataset $|\mathcal{D}|$. The total number of data points in the sample is κ and must be sufficiently big to represent the entire dataset fairly and sufficiently small to fit into the memory. Next, after determining the value of α (maximum number of data elements in a cell) on the basis of the available memory in the coordinator by using equation 3.5, the total number of cells $|\mathcal{C}| = |\mathcal{D}|/\alpha$ is estimated.

Step-2 Before constructing the in-memory *base QT*, the maximum number of data elements, $\alpha^{'}$, stored in a leaf-node of the *base QT* must be determined. Hence, $\alpha^{'}$ is computed on the basis of $|\mathcal{C}|$ (the expected total number of leaf-nodes of the final QT estimated in **Step-1**) as follows: $\alpha^{'} = \kappa/|\mathcal{C}|$, where κ is the total number of data elements in the sample. After determining $\alpha^{'}$, *base QT* is constructed in-memory over the sample data drawn in **Step-1**, and then, the sample data are discarded from the memory. Only a tree-like data structure of the *base QT* is kept in the memory for the initial partitioning of the entire dataset as explained next.

Step-3 In this step, another MapReduce job is executed to partition the entire dataset as follows: (i) In the mapping phase, the entire dataset is partitioned on the basis of the in-memory *base QT*, constructed in the previous step. Starting from the root-node of the in-memory *base QT*, each data element, **d**, traverses the tree to identify a leaf-node, to which **d** belongs. (ii) Once **d** gets into the leaf node, the Mapper class emits the key-value paired data containing a *key*, which contains the lower-left coordinates of the leaf-node, and a *value*, which is **d** itself. (iii) After the mapping process, the Reducers build several *local QTs*, which contain data elements having the same key. (iv) After constructing a *local QT*, for every non-empty leaf-node of the *local QT*, the Reducer class emits two different sets of key-value paired data in HDFS. In the first key-value set, a *key* is made up of a lower-left coordinate of a non-empty leaf node of the *local QT*, and a *value* is a list of all the data elements that belong to the leaf-node. In the second key-value set, a *key* is a lower-left coordinate of a non-empty leaf node of the *local QT*, and a *value* is the total number of data elements stored in the corresponding leaf-leaf node and the width of the leaf-node. Note that the maximum number of data elements stored in a leaf-node of a *local QT* is α, not $\alpha^{'}$.

Step-4 Recall that no data element is included in a global QT that resides in the memory of the coordinator. In each leaf-node of a global QT, it is sufficient to keep a pointer, a counter, and a width. A pointer is a lower-left coordinate of a leaf-node that is going to be used as a row-key of a corresponding row in the HBase table. In **Step-3**, for each *local QT*, a reducer class emits the key-value paired data that consist of a pointer, a counter, and a width of the non-empty leaf-nodes. Intuitively, it is straightforward to construct a global QT online by using such a list of the pointer-counter-width of each of the non-empty leaf-nodes of all the *local QTs*. .

3.6.3 Loading Data in HBase

When building a global QT over a dataset, the entire dataset is divided into several key-value paired data and stored in HDFS, but not stored in the key-value HBase table yet. As the lower-left coordinate of a leaf-node is unique and used as a key in the key-value paired data, it can be directly adapted as a row-key in the HBase table.

However, populating the large-scale key-value paired data sequentially into an HBase table has a high overhead cost; therefore, a MapReduce-based bulk loading feature is used for the parallel insertion of data into the HBase table.

3.6.4 kNN Query Processing in COWI

In COWI, as shown in Algorithm 1, a kNN query is computed as explained in the following steps: (**Step 1**) By traversing the global QT, identify the closest cell, i.e. $\mathcal{C}^* = \underset{\forall C_i \in \mathcal{C}}{\operatorname{argmin}} f(\mathbf{q}, C_i)$. Note that as a domain space, in which a dataset resides, is partitioned by a QT, a query lies within the boundaries of the closest cell $\mathcal{C}^*$; therefore, the minimum distance between $\mathcal{C}^*$ and $\mathbf{q}$ is 0: $f(\mathbf{q}, \mathcal{C}^*) = 0$, see definition 4 as proved in [69]. After identifying the closest cell, check whether there are sufficient data elements available in the corresponding row by referring to the counter stored in the closest leaf-node. If there are not enough data elements, get the second closest, the third closest, etc., until the overall sum of counters of the selected cells exceeds k. (**Step 2**) Retrieve all the data elements that reside in the corresponding rows selected in Step 1. Then, compute an initial kNN answer based on the Euclidean distance. (**Step 3**) Compute the distance, ρ, from $\mathbf{q}$ to the kth data element found in the initial kNN answer. With ρ as the radius, draw a circle centred at $\mathbf{q}$. Thereafter, select all the leaf-nodes in the QT that overlap the circle as candidates, and store them in a queue on the basis of their distance to the query in the ascending order. (**Step 4**) Remove the cell at the head of the queue, and retrieve the data elements that reside in the corresponding row of this cell. Then, compute the distance between $\mathbf{q}$ and each data element retrieved from the row. If a data element is located closer to the $\mathbf{q}$ than the data elements of the initial kNN

answer, update the initial kNN answer by adding this data element to the initial kNN answer and removing the furthest data element from the initial kNN answer. ((**Step 5**) Repeat **Step 4** until the queue that stores the candidate cells gets empty. At the end of this process, the final exact kNN is identified.

Algorithm 1: KNN processing: COWI.

```
Input: query point q, number k
Output: set A of the k nearest neighbours from q
PQ = ∅; // initialise priority queue
A = ∅; // initialise KNN list priority queue
C=∅; // initialise cell
n=0;
while n < k do
    /* the following line returns the first closest cell, the
       second closest cell, etc.  in each corresponding
       iteration                                              */
    C=getClosestCell(q);
    n+= C.getNumberOfDataElements();
end
RS = getDataFromHBase(C.getRowKey);
for each point p ∈ RS do
    A.enqueue(p, ||p − q||);
end
k^th Dis = A.getDistanceToKthElement();
PQ.enqueue(getAllCellsWithinARange(k^th Dis));
while PQ ≠ ∅ do
    C= PQ.dequeue();
    if f(q, C) > k^th Dis then
        break;
    else
        RS = getDataFromHBase(C.getRowKey);
        for each data p ∈ RS do
            if ||p − q|| < k^th Dis then
                A.removeFarthestElement();
                A.enqueue(d);
                A.updataKthElement();
                k^th Dis= A.getDistanceToKthElement();
            end
        end
    end
end
return A;
```

3.7 Second Solution: Coordinator without In-Memory Index Tree

Recall, in COWI, the size of a global index tree is determined on the basis of the available free memory in the coordinator; see section 3.5. However, the fact that the coordinator has limited memory might adversely affect the performance of COWI because it could reach a point where storing in-memory a large index tree that has a small value of α becomes impossible, particularly when indexing a very large-scale dataset. A previous study [81] reported that, in some cases, the storage cost of the index might exceed the size of the dataset. In such a case, COWI must increase α to reduce the size of the index to fit in the available free memory in the coordinator. However, increasing the value of α increases the query response time. For example, the experimental results of the uniformly generated data of Fig.3.5 show that when the value of α increases from 2,000 to 20,000 and to 200,000, the corresponding query response time increases: for $k = 10$ from 43 ms to 197 ms and to 1,394 ms; for $k = 100$ from 51 ms to 283 ms and to 1,703 ms; and for $k = 1000$ from 92 to 323 ms and to 1723 ms, respectively. This demonstrates that the query response time when $\alpha = 2000$ is 30X faster than when $\alpha = 200,000$.

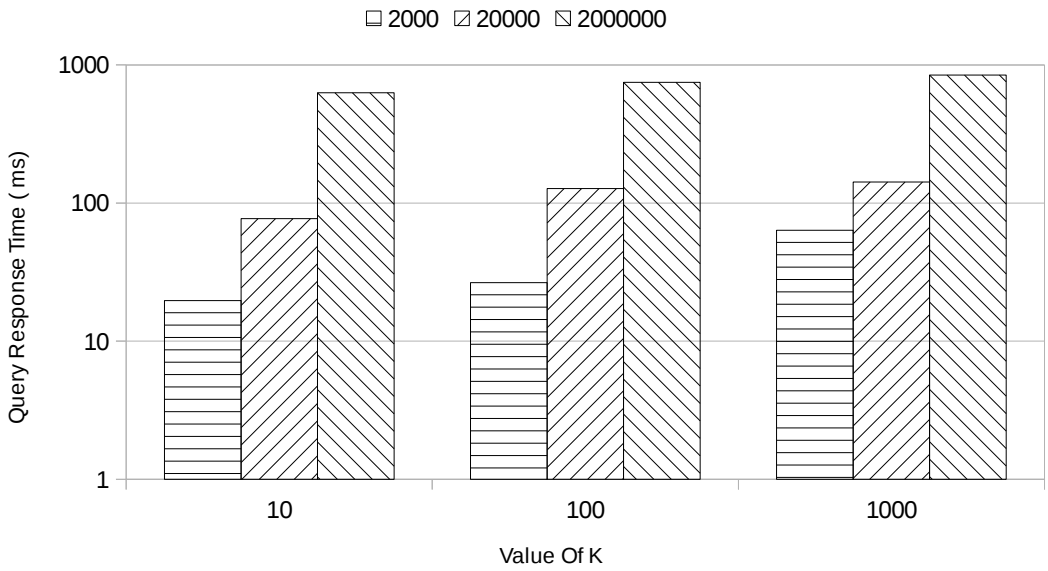


Figure 3.5: Three rows with different values of α and k.

For ensuring the high performance of a kNN query that runs over a very-large-scale dataset, the dependency of the query response time on the available free memory β must be avoided. In this section, therefore, a novel solution, Coordinator with No Index in memory (CONI), is proposed. CONI has a tiny memory footprint in the coordinator and does not take into consideration the amount of available memory in the coordinator when determining α. CONI depends only on the guidelines on how to improve the query response time when determining

α. CONI stores a part of the index tree in the key-value HBase table.

Indexing: CONI builds a QT over a very-large-scale dataset in the same way as COWI, which is explained in section 3.6.2. However, CONI determines the value of α in a different manner from that of COWI and will be explained in section 3.7.1 in detail. CONI stores a dataset and a part of the index tree in two different key-value data store HBase tables.

CONI stores the indexed dataset in an HBase table referred to as *data-table*. After a dataset is indexed and stored in a *data-table*, all the leaf-nodes of the global QT, created during the data indexing process, are indexed in turn by using another QT; CONI stores all the leaf-nodes of the global QT in an HBase table referred to as *meta-table*. Recall that a leaf-node of the global QT maintains (i) a pointer, (ii) a counter, and (iii) a width of a leaf-node of a global QT. However, only the pointers (lower-left coordinates of a leaf-node) are considered when indexing the leaf-nodes of the global QT, but in the *meta-table*, a leaf-node of the global QT is represented represented collectively as ¡pointer, counter, width¿.

At the query time, in CONI, the back-end is accessed twice: (i) the *meta-table* is accessed in order to get the row-keys of the *data-table*; and (ii) the *data-table* is accessed to retrieve the actual data elements. At the query time, to avoid a linear scan of the *meta-table*, the QT that is built over the *meta-table* can be stored in the memory of the coordinator to serve as an index. However, considering the ever-growing dataset, such a design will eventually face the same pitfall as that of COWI: the coordinator will be forced to increase α because of a lack of available memory. However, as CONI is supposed to scale well with the ever-growing data, storing the index tree in the memory of the coordinator defeats the purpose.

CONI converts the unbalanced QT, resulting from indexing the contents of the *meta-table* into a balanced QT in order to avoid storing the index tree in the memory. Intuitively, as portrayed in Figure 3.6, the leaf-nodes of a balanced QT have a uniform grid-like geometrical structure. In the coordinator, therefore, storing only the width of the cells of the uniform grid is sufficient to identify the closest cell to a query; therefore, a memory-hungry index tree is no longer required to serve as an index. This will be explained in detail in section 3.7.2.

Note that a global QT created by indexing a non-uniformly distributed dataset always has an unbalanced tree structure. Assuming that the distribution of a dataset is not uniform, both the QTs, the one used to index a dataset (aka global QT) and the one used to index the leaf-nodes of the global QT, have unbalanced tree data structures. CONI converts the later unbalanced QT to a balanced QT by adding conceptual nodes, which are nodes connected by the dotted lines in Figure 3.6. Hence, the resulting balanced QT has two types of leaf-nodes: the actual leaf-nodes and the conceptual leaf-nodes. Each actual leaf-node has a pointer to a unique row in the *meta-table*. However, several conceptual nodes point to a single row in the *meta-table*. The conceptual leaf-nodes copy a pointer inherited from the actual parent node: an actual parent node is a leaf-node of the unbalanced QT from which the conceptual nodes are

extended.

As CONI has a tiny memory footprint in the coordinator, we can choose the value of α freely without any restriction imposed by β. Thus, for any very-large-scale dataset, CONI scales well without sacrificing the query performance.

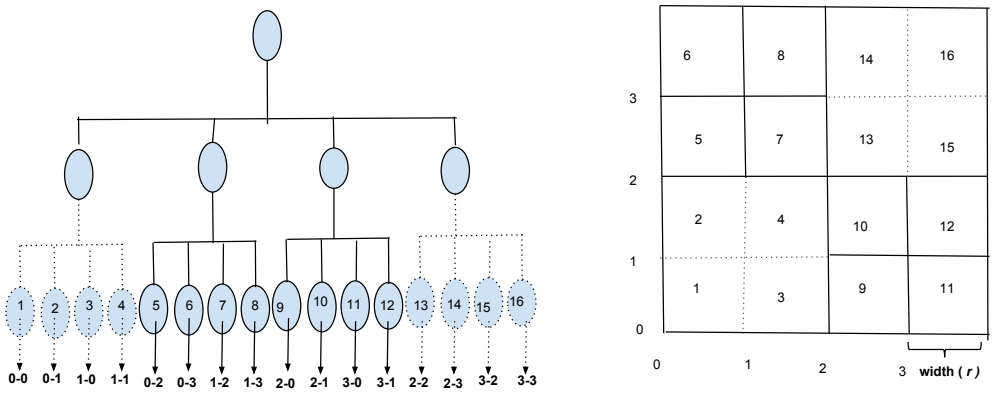


Figure 3.6: Conceptual nodes are connected by dotted lines.

In summary, CONI:

- stores a small number of data elements, α, in a *data-table* row irrespective of β and the size of the dataset;
- stores the main index in *meta-table*; and
- has a minimal memory footprint.

Furthermore, note that one can argue that it would be better to store the contents of the *meta-table* at the coordinator instead of in a key-value HBase table. Such an argument has more weight w.r.t the expedited access to the index table, because accessing a local disk is relatively cheaper than accessing a distributed key-value data store. However, it contradicts with several principles of big data systems. CONI depends entirely on the index table for query processing; therefore, the contents of the *meta-table* must be highly available, easily recoverable, and easily accessible. By storing the index table in HBase, CONI benefits from such tenets. Thus, CONI is a robust solution albeit at the additional overhead cost of accessing the distributed key-value datastore.

3.7.1 Determination of the Value of α

CONI determines the value of α on the basis of two fundamental principles. According to the first principle, in HBase, retrieving k rows, each containing one data element, is more expensive than retrieving one row that contains k data elements: retrieving fewer 'fatter' rows is considerably more efficient than retrieving many 'thin' rows [45]. The second principle states that retrieving a row that contains $\alpha >> k$ has a high query response time, as shown in

Figure 3.5. Therefore, the value of α must be neither two few nor too much as compared to k. Hypothetically, the value of α needs to be sufficiently large to accommodate a reasonable expectation that most kNN queries will be included in a single data cell and be sufficiently small to reduce irrelevant data access.

Unfortunately, the value of k is unknown in advance. However, a reasonable value of k can be estimated; typical users are interested in k=1 to k=1000 as mentioned in [66]. CONI determines the value of α on the basis of the most frequently used maximum value of k. In the experiments discussed in this chapter, α was set to 2000.

3.7.2 Uniform Grid as Index to the Meta-Table

Recall that the leaf-nodes of a balanced QT are squares with equal size and can be represented by a uniform grid. A lower-left coordinate of the balanced QT points to a row in *meta-table*. Thus, the coordinator has to find the lower-left coordinates of the closest leaf-node, in which a query lies, on the basis of only the information stored in the memory, which is the width of the leaf-nodes.

During query execution, CONI finds the lower-left coordinates of the closest leaf-node by applying definition 11 for each dimension separately. Each dimension of a uniform grid, created by a balanced QT, is divided into several intervals, each of which has the following: (i) a starting point, (ii) an ending point, and (iii) equal width with the other intervals; note that an interval is one of the sides of a leaf-node of a balanced QT. When computing the lower-left coordinates of the closest leaf-node, it is sufficient to find the starting points of the intervals (sides) of a leaf-node by considering one dimension at a time, as shown in the following example.

Example: Consider a random point (2.05, 1.8) that lies in the 10th cell in Figure 3.6. Assume that the lower-left coordinates of the 10th cell, in which the random point resides, is not known in advance and finding such coordinates is of interest. By applying definition 11 ($\lfloor \frac{q}{r} \rfloor \cdot r$, where r is the width of the leaf-nodes and q is the value of a given coordinate), the lower-left coordinates of the cell can be computed as follows: given $r = 1$, the value of the left-most x-coordinate of the cell is $\lfloor (2.05/1) \rfloor \cdot 1 = 2$ and the lower-left value of the y-coordinate is $\lfloor (1.8/1) \rfloor \cdot 1 = 1$; hence, the lower-left coordinates of the cell are (2,1), which are the same as the lower-left coordinates of the 10th cell shown in figure 3.6.

3.7.3 kNN Query Processing in CONI

In CONI, a kNN query is processed as shown in algorithm 2 and explained below:

1. Get the closest cell to a query $\mathbf{q}$ from the corresponding cells of the meta-data table, the ith dimension of the lower-left coordinates of the winner cells is computed using $\lfloor (q_i \div r) \rfloor \cdot r$. If there are insufficient points in the rows of the data-table, whose key-rows lie in the winner meta-table cell, get the second closest meta-cell from the neighbouring cells of the winner meta-cell, get the third closest meta-cell, and so on, until the total number of points that reside within these cells exceeds k.
2. On the basis of definition 4, compute $f(\mathbf{q}, C_i)$, which is the distance between query $\mathbf{q}$ and every cell that corresponds to the data-table, which is retrieved in Step 1, and sort these cells in the ascending order on the basis of their distance from $\mathbf{q}$.
3. Get the shortlisted candidates by selecting the closest top t cells that contain k or more points in total from the list obtained in Step 2. Retrieve all the points that reside in the furthest row from the shortlisted candidates, and compute the initial kNN answer on the basis of the Euclidean distance.
4. Use the Euclidean distance from $\mathbf{q}$ to the k-th point in the initial kNN answer as a radius in order to draw a circle centred at $\mathbf{q}$. Then, retrieve all the points that reside in the data-rows, whose keys are in turn stored in the meta-rows, which overlap the circle.
5. Check whether there are points that are retrieved in Steps 3 and 4 and are located at a closer distance to $\mathbf{q}$ than the points selected as members of the initial kNN answer. If so, add them to the kNN answer and remove the furthest points. At the end, the kNN answer will contain the correct set of points.

3.8 Updates

3.9 Updates

After a dataset is indexed and stored in the HBase table, new data elements might be inserted into the database without redoing the expensive indexing process. In such a case, the rows of the HBase table might contain more than α data elements; however, to maintain high kNN query performance, rows that contain a considerably large number of data elements must split without re-indexing the entire dataset.

In large-scale data processing, neither re-indexing an entire dataset nor splitting rows that contain more than α data elements is cheap. The indexing process might take hours if not days; similarly, the row splitting process has a high overhead cost because of the memory lock and the requirement of disk and network I/Os. To tackle this issue, in this part of the thesis, a *lazy dynamic partitioning technique* is proposed to carry out the data update process efficiently. Contrary to the QT leaf-node splitting criterion, in a *lazy dynamic partitioning*

Algorithm 2: KNN processing: CONI.

```
Input: query point q, number k
Output: set A of the k nearest neighbours from q
PQ = ∅; // initialise priority queue that contains cells within a given range
RC = ∅; // initialise priority queue that contains candidate cells
A = ∅; // initialise KNN priority queue
C=∅; // initialise cell
m=∅; // initialise meta-cell
n=0;
while n < k do
    /* the following line returns the first closest meta-cell,
       second closest meta-cell, etc.  in each corresponding
       iteration                                              */
    m=getClosestMetaCell(q);
    n+= m.getNumberOfDataElements();
end
for each cell c ∈ m do
    RC.enqueue(c, f(c,q));
end
while RC ≠ ∅ do
    C=RC.dequeue();
    n+= C.getNumberOfDataElements();
    if n >= k then
        break;
    end
end
RS = getDataFromHBase(C.getRowKey);
for each data d ∈ RS do
    A.enqueue(d, ||d − q||));
end
k^th Dis = A.getDistanceToKthElement();
PQ.enqueue(getAllCellsWithinARange(k^th Dis));
while PQ ≠ ∅ do
    C= PQ.dequeue();
    if f(c,q) > k^th Dis then
        break;
    else
        RS = getDataFromHBase(C.getRowKey);
        for each data d ∈ RS do
            if Dis(d,q) < k^th Dis then
                A.removeFarthestElement();
                A.enqueue(d);
                A.updataKthElement();
                k^th Dis=A.getDistanceToKthElement();
            end
        end
    end
end
return A;
```

method, the decision to split a row does not depend on α, but on the following weight cost function:

$$u = \gamma \times |C_i|, \tag{3.6}$$

where γ refers to the total number of times a row has been queried, $|\mathcal{C}_i|$ is the total number of data elements stored in the row, and u is the weight' of the row. In a *lazy dynamic partitioning* method, a row splits when its weight u exceeds the user-predefined threshold. The rationale behind the lazy dynamic partitioning technique is to split the frequently queried rows more aggressively than the less queried rows. Tolerating less queried rows to swell up does not affect the performance of most of the kNN queries.

However, in COWI, the row splitting process becomes more convoluted, particularly when there is insufficient memory available in the coordinator to accommodate the newly created leaf-nodes. Consider that a leaf-node that is about to split is called an *overloaded leaf-node*. Conceptually, an overloaded leaf-node splits into up to x leaf nodes, where x is the maximum number of child nodes of the QT algorithm. After a continuous operation, in essence, a subtree rooted at the *overloaded leaf-node* is created; however, if the original QT is designed to occupy all of the available memory at the coordinator, there is no free room available for the newly created subtree to squeeze into the memory. Therefore, borrowing from the CONI approach (§3.7), one can store the contents of the newly created subtree in a key-value HBase table. The contents of the HBase table offer the same functionality as if the original overloaded leaves were split and stored in the memory.

In particular, the process is as follows: Because of the lack of free memory space at the coordinator, no additional subtree is annexed to the original index tree; the original QT always remains intact in the memory. When a row in the HBase table splits, the corresponding leaf-node that resides in the memory should point to a row in the HBase table, in which the leaf-nodes of the newly created subtree reside.

Recall that a leaf-node is represented by $<$ pointer, counter, width $>$. Each leaf-node of the new sub-tree, therefore, contains (i) a pointer that points to the corresponding new row in the data-table (in which the actual indexed data elements reside), (ii) a counter that quantifies the total number of data elements stored in the newly created row, and (iii) a width that contains the side-length of the new leaf-node that is supposed to be stored in the memory.

3.10 Performance Evaluation

3.10.1 Experimental Set-up

Comprehensive experiments were carried out to study the performance of the proposed approaches (COWI and CONI) as compared to that of the state-of-the-art methods such as

SHadoop (SH) [35], the best solution from the Hadoop ecosystem, and SIMBA [87], the best solution from the Spark ecosystem. In the experiments, owing to the authors, the publicly available codes of Simba and Hadoop were used. These experiments were run on a five-node cluster; each node was a Dell R720 server with four Intel Xeon(R) CPUs (eight cores each), 64 GB RAM, and approximately 4 TB of disk space.

Datasets

Several datasetsof different sizes are used in the experiments; the datasets contained two-dimensional data points.

Several two-dimensional datasets of different size were used in the experiments. Ten synthetic datasets of different sizes are generated as follows: (1) a dataset that contained circa 600 million points with a total size of 20GB, (2) a dataset with 1 billion points and a total size of 35 GB, (3) a dataset that contains 7 billion points with a total size of 250GB, (4) dataset that contains around 29 billion points with a total size of 1TB, and (5) a dataset that contains circa 100 billion points with a total size of 3.5TB - this is the largest dataset that the cluster can accommodate. These datasets were generated in an area of $1M \cdot 1M$ units, and all the points were generated on the basis of the uniform distribution as [35]. Additional five datasets of the same size were also generated on the basis of a multi-modal distribution. Each of these dataset was generated on the basis of the following four Gaussian mixture models, each of which has 0.25 mixing weight: (i) mean $[450, 250]^T$ and covariance $\begin{pmatrix} 150 & 0.9 \\ 0.9 & 200 \end{pmatrix}$; (ii) mean $[250, 500]^T$ and covariance $\begin{pmatrix} 150 & 0.9 \\ 0.9 & 200 \end{pmatrix}$; (iii) mean $[550, 250]^T$ and covariance $\begin{pmatrix} 150 & -0.9 \\ -0.9 & 200 \end{pmatrix}$; and (iv) mean $[500, 500]^T$ and covariance $\begin{pmatrix} 150 & 0.9 \\ 0.9 & 200 \end{pmatrix}$;

As in SH, 10^4 query points were randomly selected from the input files and issued over the datasets for different values of $k \in \{10, 100, 1000\}$.

Performance metrics

Each method executed all queries sequentially, and then, the average query response time, measured in milliseconds, was reported. Moreover, three other qualitative methods were considered: (i) average number of rows (cells) retrieved per query, (ii) average number of data points accessed per query, and (iii) time t required at the coordinator to process the contents of the retrieved cells (rows) and produce the final kNN answer. These three measures were essential showcasing the scalability of the proposed design.

SHadoop uses two multi-dimensional index approaches, namely grid and R-Tree. The R-Tree is supposed to index a non-uniformly distributed data by using the publically available code of SHadoop. However, several attempts have been made to index the non-uniformly distribute data using R-tree; unfortunately, the MR process hanged repeatedly. Consequently, the proposed approaches (COWI and CONI) are compared to SHadoop with a grid index for the uniformly distributed datasets. Note that as the SHadoop authors pointed out, the grid-based indexing approach performs well in a uniformly distributed dataset.

Additionally, as found in the source code (not reported in the scientific paper), SHadoop has two distinct methods of executing kNN queries: (i) with MR and (ii) without MR. The former method launches an MR job when executing kNN queries, but the latter method retrieves files directly from the HDFS without the MR jobs. To be fair, the proposed approaches were compared against both variants of SHadoop: SH with MR (SH-MR) and SH with HDFS, without MR, (SH-HDFS).

Similarly, the publically available code of Simba was used in the experiments. However, the built-in indexing method of Simba can not index datasets bigger than 1 billion points; Simba repeatedly crashed while indexing bigger datasets. Therefore, Simba was compared to COWI and CONI using only the 25GB and 35GB datasets, each of which contains 600 million and 1 billion data points respectively. Note that this is sufficient to showcase the superiority of the proposed approach against Simba; even for smaller datasets, Simba has up to two orders of magnitude slower query response time than CONI/COWI (even when the latter was run on the bigger datasets).

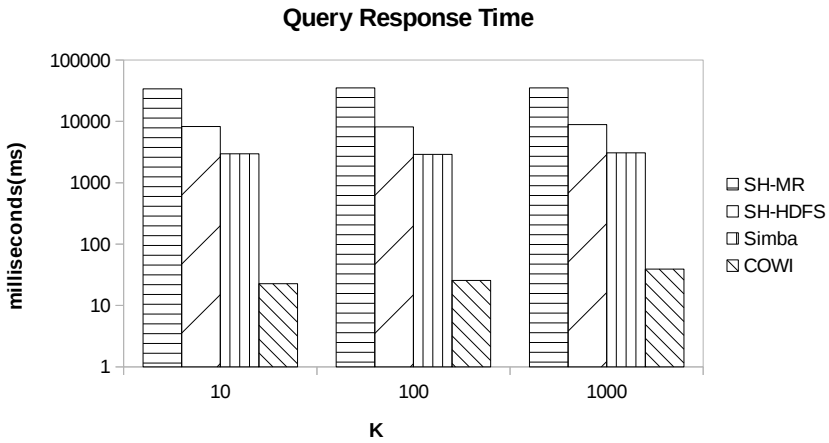


Figure 3.7: Dataset: 600 Million data points (20GB)

3.10.2 Performance Assessment

In Figures 3.7 and 3.8, the kNN query response time, on the 20-GB and 35-GB datasets, of COWI (§3.6), SH-HDFS, and Simba are depicted. The kNN query response time was

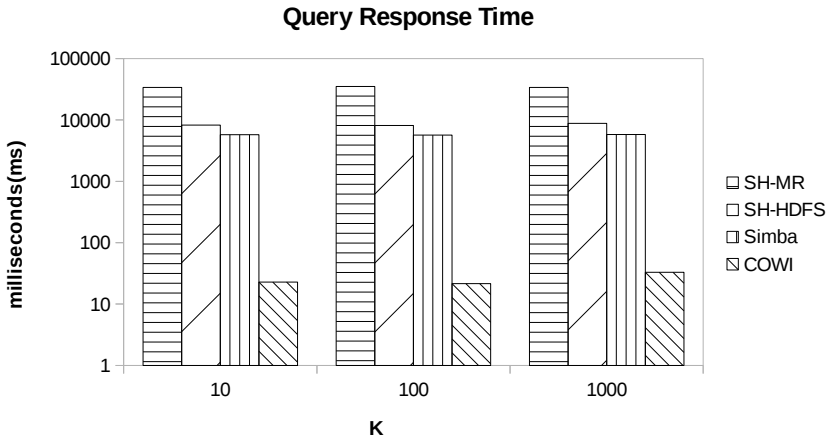


Figure 3.8: Dataset: 1 Billion data points (35GB)

measured in milliseconds (ms) for different values of k. The query response time of COWI varied from 22 ms to 32 ms, whereas the query response time of SH-MR varied from 34,000 ms to 35,000 ms, that of SH-HDFS varied from 8,100 ms to 8900 ms, and that of Simba varied from 3,000 ms to 5,000 ms. The performance results of the Simba and SH-MR approaches were in line with the results reported by the authors of Simba, with respect to the relative performance of SH-MR and Simba. COWI achieved query performance gains of up to two orders of magnitude as compared to Simba and achieved query performance gains of more than two orders of magnitude as compared to both the implementations of SH.

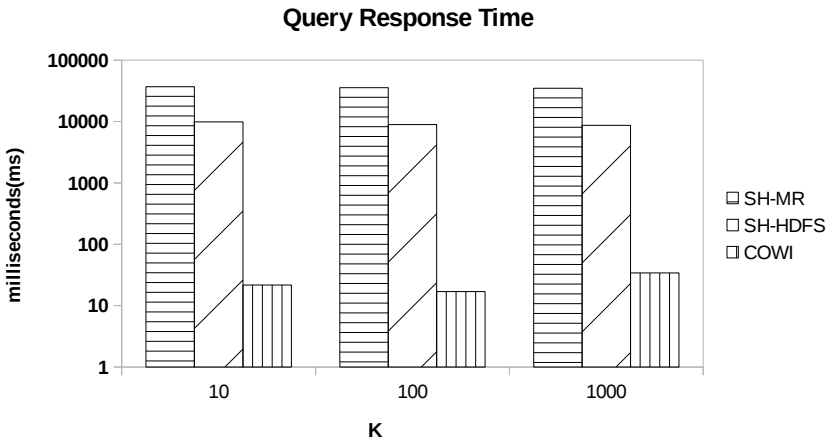


Figure 3.9: Dataset: 7.3 billion data points (250 GB)

The same conclusions held for the 250-GB and 1-TB datasets, shown in Figure 3.9 and 3.10, respectively. Note that all the approaches showed excellent scalability: a very small increase in the query response time was observed for an approximately two orders of magnitude increase in the size of the dataset. Moreover, the same conclusions held across the different datasets generated on the basis of the multi-modal distributions, as shown in Figure 3.12; recall that as SHadoop and Simba failed to index the datasets generated by the multi-modal distribution, only the results for COWI and CONI are presented.

To further stress-test the proposed approaches and to study how the size of a cell affected the query response times, a 3.5-TB dataset was generated. In COWI, the maximum number of data elements stored in a cell was set to 10^6l, and in CONI, the maximum number of data elements stored in a cell was set to 2,000. As shown in Fig. 3.11, while the query processing time of COWI increased by more than 10X (to between 526 ms and 595.87 ms), in the case of CONI, the query processing time remained within the range of 91.4 ms to 185 ms. Thus, CONI continued to offer gains of several orders of magnitude and showed additional benefits (in addition to those stemming from not requiring memory resources at the coordinator).

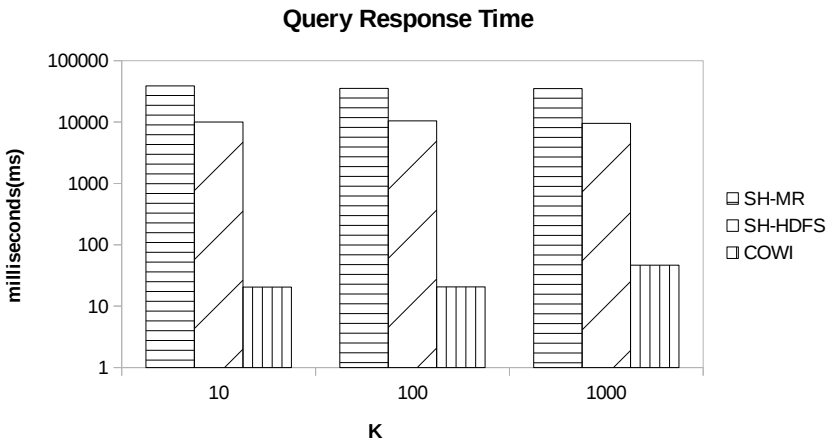


Figure 3.10: Dataset: 29.1 billion data points (1 TB).

Note that the above experiments showcased that the core design choices for the proposed approaches paid off significantly. Unlike Hadoop-MR-based and Spark-based approaches (such as SH, Aqwa, GeoSprk, and Simba), the proposed methods managed to store few data elements per cell. Further, unlike Simba, SH, or Aqwa, the proposed methods improved the query response time considerably by ensuring access to only small but relevant data elements. The above experiments and comparisons quantified the relevant costs associated with each design choice.

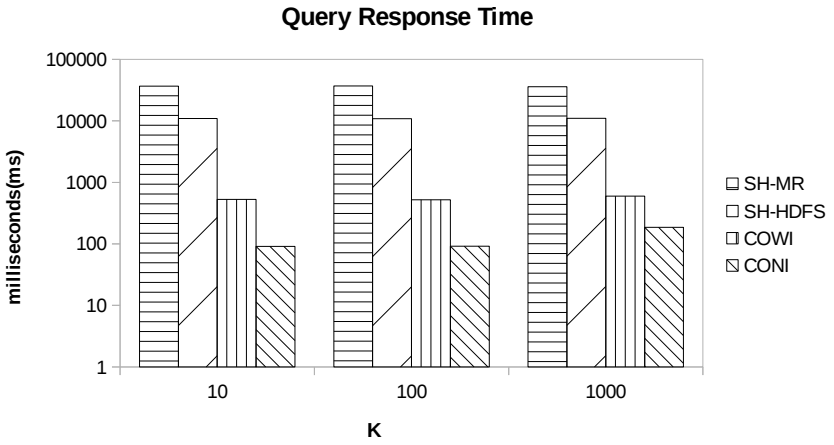


Figure 3.11: Dataset: 100 billion data points (3.5 TB).

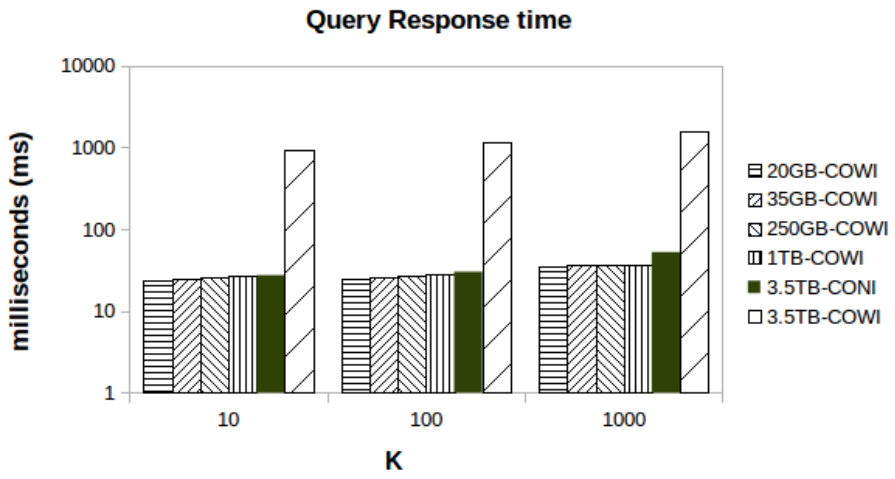


Figure 3.12: Dataset: Multi-modal distribution.

In order to evaluate the scalability of the proposed approaches, the average accessed number of rows (cells) and number of data elements per query were also measured. This is fundamental for any coordinator-based approach, as the network bandwidth of the coordinator can be saturated and the same holds for the CPU processing data elements retrieved from the data store.

As shown in Figure 3.13, the average number of rows accessed by both the approaches remained almost the same, even when the size of a dataset increased. The average number of rows accessed by both the approaches only increases with larger values of k, as expected. In our smallest dataset (20 GB), when the value of $k = 10$ and $\alpha = 2000$, COWI accesses on average 1.17 rows (cells) per query. For the same values of k and α, when the dataset size increased to 250 GB and 1 TB, on average only 1.17 and 1.176 rows per query were accessed, respectively. Thus, COWI scales very well in terms of the average number of rows accessed per query with increasing dataset sizes. When the value of k increased to 1000 and $\alpha = 2000$, on average 2.8, 2.89, and 2.909 rows were accessed per query for dataset sizes of 20 GB, 250 GB, and 1 TB, respectively.

Last but not the least, when the dataset size was 3.5 TB and $\alpha = 2000$, CONI accessed on average 2.16, 2.73, and 3.29 rows for k equal to 10,100 and 1,000, respectively. In contrast, COWI for $\alpha = 100000$ accessed on average 1.01, 1.07, and 1.24 rows when k was 10, 100, and 1000, respectively. On average, COWI accessed fewer rows than CONI in the 3.5-TB dataset. This was because CONI had to access more rows of the meta-table in order to identify the closest cell, $\mathcal{C}^*$. With this result, it was easy to see and quantify the tensions between CONI and COWI: CONI could reduce the value for α, but at the expense of needing to access additional meta-table rows from HBase. In contrast, COWI required no additional HBase accesses, as the results above showed, but had to use a considerably higher value for

α. In a similar venue, the results obtained from the datasets generated by the multi-modal

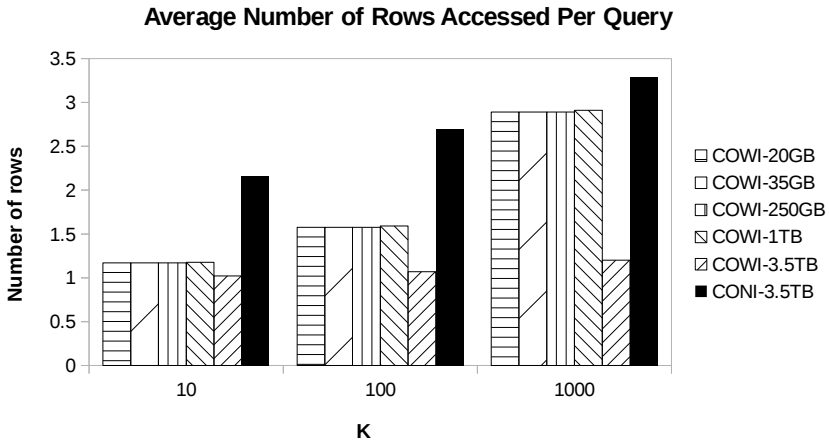


Figure 3.13: Average number of rows accessed per query.

distribution were similar to those of uniformly distributed datasets, as shown in Figure 3.14.

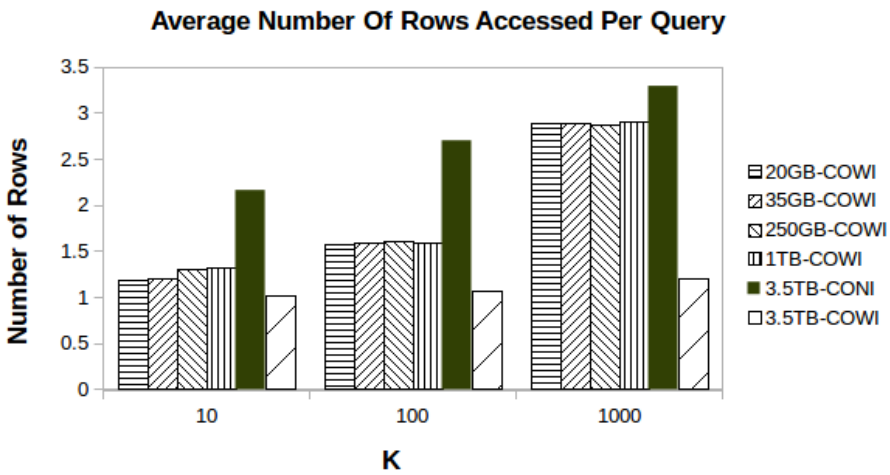


Figure 3.14: Average number of rows accessed per query for multi-modal datasets.

Figure 3.13 shows that in both COWI and CONI, the average number of rows accessed per query increased by a negligible value when the dataset increased significantly; similarly, the average number of data elements accessed per query was not significantly affected by the size of the dataset. As illustrated in Figure 3.15, for k=10 and $\alpha = 2000$, COWI accessed on overage 2,340, 2,340, and 2,339 data elements when the dataset size was 20 GB, 250 GB, and 1 TB, respectively. Moreover, CONI accessed on average 2,471 data elements per query when $k = 10$, and $\alpha = 2,000$ for the 3.5-TB dataset; but for the same dataset (3.5 TB), COWI accessed 99,312 data elements for $k = 10$ because α was set to a considerably higher value, i.e. $\alpha = 100,000$. This demonstrated that when $\alpha >> k$, many unnecessary

data elements were accessed, and as such, the query response time was adversely affected, as depicted in Figure 3.11. Fundamentally, the results in Figure 3.15 show that with COWI or CONI, on average, relatively, the same number of data elements per query were accessed (when $\alpha = 2,000$) across widely varying dataset sizes. The same results were noted for the datasets generated by the multi-modal distribution, as shown in Figure 3.16.

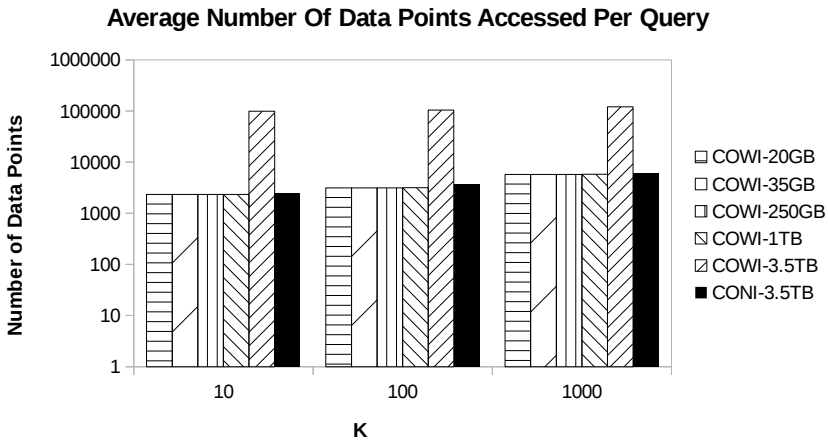


Figure 3.15: Average number of data points accessed per query.

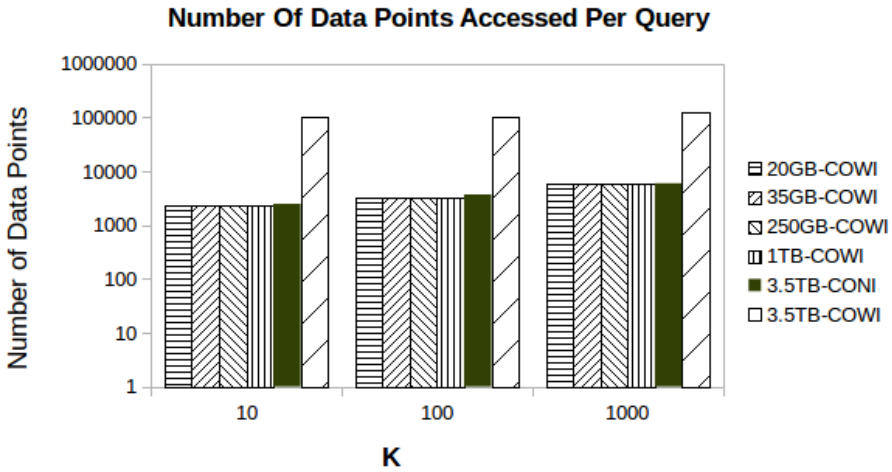


Figure 3.16: Average number of data points accessed per query for multi-modal datasets.

Finally, in addition to the above two qualitative measures, measuring the query processing time at the coordinator (needed to process the retrieved data elements) is a fundamental aspect to scalability. Figures 3.17 and 3.18 show that for the uniform and multi-modal datasets, the dataset size had a very small effect on the time that the coordinator had to devote for data crunching. Note that, as expected, the processing time at the coordinator with COWI increased significantly with the largest dataset, as α assumed greater values and this led to a very large number of points that had to be (communicated to and) processed by the coordinator to produce the final query answer.

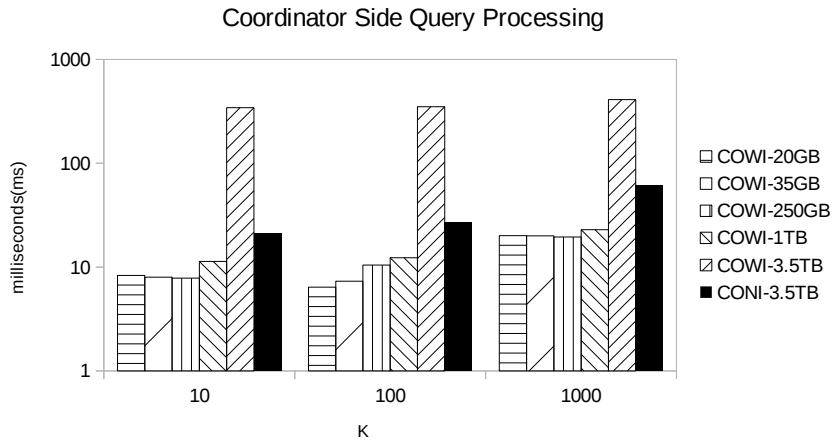


Figure 3.17: Coordinator processing time.

Figure 3.18: Coordinator processing time for multi-modal datasets.

3.11 Conclusions

In this work, a novel approach for processing kNN queries was proposed. The principal masterpiece of this approach rested upon two key features: First, in contrast to the state-of-the-art approaches that operate either on Hadoop or Spark systems, the proposed approach was a coordinator-based parallel/distributed kNN query processing method. In the proposed approach, scalability was not achieved at the expense of query processing efficiency; this was a distinct and important phenomenon of the proposed solution. As it was shown that kNN query had to and could be processed in a matter of few tens of milliseconds and not several (tens of) seconds, a Hadoop/MR or Spark based solution unnecessarily sacrificed the query processing efficiency to achieve scalability. Second, in order to access only the relevant data elements, the state-of-the-art methods of data organisation, of data storage, and of data indexing were carefully revisited. Why should an algorithm retrieve from storage, communicate, and process millions of other data items, when processing a 10NN query? As the impact of the cell size on key scalability factors, such as the size of the available memory at the coordinator, were studied thoughtfully, two approaches: COWI and CONI were proposed. The results of extensive experiments conducted to compare the performance of COWI and CONI against the state-of-the-art methods, SHadoop and Simba, demonstrated that the proposed approaches achieved up to three orders of magnitude lower query processing times; moreover, the experimental results showed that the overall query processing times scaled exceptionally well with dataset sizes. The number of cells and data elements retrieved, communicated and processed were carefully evaluated, and hence the results further substantiated the scalability of the proposed approaches.

Chapter 4

Scaling kNN Queries via Probabilistic Data Space Transformations

4.1 Introduction

Due to the ever-growing data volumes and complexity, the traditional off-the-shelf data processing tools become inadequate to pace with the fast-evolving data. As explained earlier, several parallel/distributed processing frameworks, e.g., Hadoop-MapReduce [31] and Spark [92], have evolved to deal with the issue. However, as such frameworks scan the whole dataset (even in parallel), can not process kNN queries efficiently. To this end, several solutions have been proposed for efficient processing of kNN queries; the current state of the art consists of three main contenders: (i) SHadoop [35], the best from the Hadoop system; (ii) Simba [87], the best from the Spark system; and (iii) COWI/CONI, which are introduced in the previous chapter.

SHadoop and Simba define relatively large data-blocks because the lower size of a data-block is determined based on the setting in which the frameworks operate. For instance, as the minimum block size of the Hadoop distributed file system (HDFS) is 128MB, the minimum size of a data-block in SHadoop is 128MB. However, during query execution, accessing such large data-blocks hurts the performance of a kNN query because retrieving and processing too many unnecessary data elements from a distributed file system has high CPU and disk/network I/Os overhead cost. To this end, COWI/CONI are proposed in the previous chapter to avoid such an unnecessary overhead cost when executing kNN queries. COWI/CONI define a small-sized data-blocks a.k.a cells; thus, a minimal but relevant number of data elements are accessed at query time. Consequently, COWI/CONI achieve up to three orders of magnitude performance gain comparing against SHadoop and Simba.

All the methods mentioned above rely on tree-based multi-dimensional index approaches for

indexing large-scale datasets. However, as Tree-based multi-dimensional index approaches were initially designed to index small datasets that reside in centralised systems, do not scale well in distributed systems that are designed for processing large-scale datasets.

Moreover, tree-based approaches are not only used to divide a dataset into several partitions but also build, in order to serve as an index, a tree-like data structure that supposed to be stored in the main memory of the coordinatore - a centralised machine that supposed to store the index tree. However, in the big data environment, the size(in bytes) of the tree-like data structure (index) sometimes can surpass the size of the dataset itself [81]; and the problem exacerbates when a large-scale dataset is partitioned into small cells. As it inevitably creates a larger-sized index tree, which very likely might not fit in a designated memory assigned in the coordinator.

Motivation Example, consider a 1PB dataset that contains two-dimensional data. Assume one opts to store only 2000 data points per cell in order to achieve high kNN query performance; thus, a minimum size of all the leaf-nodes of an index tree that supposed to be stored in memory is $a(2000 \cdot l) = 1PB \approx 500GB$, where a is the size of a data point in bytes and l is the total number of leaf nodes. This indicates that considerably larger than 500GB (as this does not include the overhead size of the nodes and size of the none leaf-nodes) free memory should be available at the coordinator; for example, if the index is a binary and balanced tree, more than 1TB memory is needed. If one wants to reduce the size of the index tree, there are two known solutions. A simple solution for reducing the size of the index tree is to increase the size of data-blocks (partitions); but as explained in the previous chapter, having large data-blocks decreases kNN query performance. In addition to that, when a size of data-blocks increases with dataset size, scalability of the kNN query compromises significantly. In the previous chapter, to avoid the tension between reducing the size of the index tree and high kNN query performance, a solution coined CONI was proposed. However, CONI has two notable drawbacks: (i) at query time, accessing an index stored in HBase table has high latency compared to an index stored in memory, and (ii) CONI needs to replicate parts of the index contents in order to create a balanced tree; so CONI may have a larger disk footprint.

In this chapter, therefore, a novel perspective for organising datasets that alleviates the need for any traditional tree-based index data structure is proposed. The main contribution of this work stem from the following fundamental observations:

Observation 1

By employing a simple uniform-grid indexing method, a dataset generated by a joint uniform distribution can be partitioned into equal-sized cells, each of which contains circa the same number of data elements. Since those cells have equal size, no need for the memory-hungry tree-like data structure to be stored either in memory or disk in order to serve as an index.

All Closest cells to a query can be identified by simple arithmetic: a simple division of a query's coordinates over a width of the cells. Accordingly, a uniform-grid index approach has an exceptionally strong performance and scalability when the distribution of a dataset is uniform.

Observation 2

The performance of uniform-gird indexing approach deteriorates as the distribution of the dataset deviates from a uniform distribution. The more skewed the distribution of the dataset is the weaker performance of the uniform-gird method.

Real-world datasets rarely have Uniform distribution; so, if the uniform-grid indexing approach is applied to index non-uniformly distributed real-world dataset, the resulting cells contain an unpredictable number of data elements per cell. Few cells can contain a large number of data elements while a significant amount of cells might be empty or near empty; hence, query processing time prolonged as retrieving and processing cells that contain a large number of data elements is expensive.

Observation 3

Fortunately, when the Random Variables (RVs) of a dataset are statistically independent of each other, the distribution of the dataset can be transformed into a joint uniform distribution based on well-known statistical methods, coined Independence Copula [40]. On the other hand, when there is dependency between the RVs of a dataset, but the joint and marginal distributions of such RVs are members of elliptical distributions [19], then removing correlation between the RVs implies independence [67].

Observation 4

Unfortunately, as the distributions of many real-world datasets do not belong to the family of elliptical distributions, removing correlation between the RVs of such datasets does not yield statistically independent RVs. However, luckily, according to [58], any arbitrary random distribution of a dataset can be approximated by a finite Gaussian Mixture Models (GMM) to arbitrary accuracy. Thus, by clustering a dataset with the right number of components, GMMs can approximate the unknown underlying probability density function of a dataset [58]. As each component (cluster) of the GMM has a multivariate (mv) Gaussian distribution (and thus elliptical), removing the correlation between RVs of each component implies *Independence* and hence using the Independence Copula, the distribution of each component can be transformed to a joint uniform distribution.

Based on the above observations, in this chapter, a dataset is assumed to have a distribution that belongs to the elliptical distributions or that can be approximated to Gaussian distribution using GMM with reasonable accuracy. The fundamental idea is that by transforming a dataset to a uniformly distributed dataset, the simple uniform-grid indexing approach can be build over the transformed dataset. Thus no need to have the memory-hungry tree-based index either in memory or disk and no need to sacrifice neither query performance nor is system scalability when processing kNN queries over a large-scale dataset.

4.2 Contribution

The salient contributions are:

- A novel approach for organising mv datasets, based on a Space Transformation Organisation Structure (STOS), which facilitates kNN query processing as if the underlying datasets are uniformly distributed. This approach ensures:
  - Extremely low memory footprint, several orders of magnitude smaller than index-based approaches.
  - A memory footprint that is practically independent of the dataset size and the number of data points per grid cell – a fact that ensures scalability.
  - Fast STOS computation time, that is smaller than traditional index building times, by several orders of magnitude.
  - Easy and fast recoverability, as the minute size of STOS allows for several copies of it to be distributed, thus allowing the system to be up and running quickly after failures.
  - Query processing similar or better than traditional (tree-based) indexing approaches.
- The above claims are substantiated using extensive experiments on real and synthetic datasets, across various dimensionalities.

This part of the research is published on the International Conference on Big Data and **was awarded Best Student Paper**

- A. S. Cahsai, C. Anagnostopoulos, N. Ntarmos, and P. Triantafillou (2018), *Revisiting Exact kNN Query Processing with Probabilistic Data Space Transformations*, IEEE International Conference on Big Data.

The rest of the chapter is organised as follows: Section 4.3 reviews background and related work, while Section 4.4 explains the preliminary, Section 4.4 presents the proposed method, Section 4.5 the proposed method, and Sections 4.6 and 4.7 report on implementation details and experimental evaluation, respectively. Finally, Section 4.8 concludes the chapter.

4.3 Background & Related Work

Traditional multi-dimensional indexing approaches that are initially designed to index small datasets in a centralised system are adapted for indexing large-scale datasets in parallel/distributed systems. In the literature, several adaptation strategies have been proposed. In this section, such methods are briefly summarised, and their strengths and weaknesses are discussed.

4.3.1 Related Work

Hadoop/MR Approaches

Several solutions have been proposed for processing kNN queries in the Hadoop system, e.g. [21, 5, 7, 35]; for the sake of simplicity, [21] and the most recent, best-performing method coined SHadoop [35] are discussed in the following paragraphs.

SHadoop builds global and local multi-dimensional indexes. During query execution, irrelevant data partitions are pruned out using the global index, whereas unnecessary data in a partition are filtered out using a local index. Each partition has its own local index.

SHadoop builds an in-memory R-tree based on a small data sample drawn uniformly at random from the input data. The default size of the sample is 1% of the input data with a maximum size of 100 MB in order to ensure that it fits in the memory. SHadoop bulk-loads the sample data to the in-memory R-tree by using a sort-tile-recursive (STR) packaging method [54]. Then, the entire dataset is partitioned on the basis of the in-memory R-tree, and another R-Tee is built over all the partitions in order to serve as a global index.

However, according to a recent survey [73], such data partitioning methods work fine for uniformly distributed data but do not work well for non-uniformly distributed data. A case in point, in the previous chapter, indexing non-uniform datasets using the publicly available SHadoop implementation was not possible, even for medium sizes (such as 250 GB). This was due to a small number of processes (reducers) hanging and failing to finish: a clear sign of poor load balancing lurking with non-uniform data distributions.

The skewness (load imbalance) of the data partitioning process in STR improves as the sample size increases; for example, STR achieves a near-perfect load balancing when the size of the sample is as big as the entire dataset [34]. However, having a considerably large sample might increase the index building time and/or might not fit into the memory allocated to the container (in which the mappers or executors run). Choosing the right sample size depends on the size and distribution of a dataset and is not an easy task; for instance, for some datasets, a small sample can be sufficiently good to index a dataset, but for the others, a bigger sample is required. Repeating the indexing process several timeswith different sample sizesbefore successfully completing the indexing process wastes considerable time and

resources, as indexing is a time-consuming and resource-demanding process. Therefore, the indexing mechanism of SHadoop has sacrificed performance and/or robustness by setting the default maximum size of the sample to 100 MB.

In contrast, another MR-based method that uses a clustering technique for partitioning a dataset was proposed by [21]. The X-means uses an iterative clustering algorithm with Bayesian information criteria (BIC) to cluster a dataset according to the Gaussian distribution. However, during the clustering process, unlike our method that scans the entire dataset only once, [21] does multiple scans over the entire dataset by using several MapReduce jobs iteratively. Thus, [21] has high data indexing time.

Spark Approaches

The state-of-the-art method of kNN processing in the Spark system is SIMBA [**?**], according to the authors. SIMBA extends Spark SQL by adding global and local indexes to prune out irrelevant partitions and data elements, respectively. SIMBA accesses relatively small amounts of data as compared to Spark SQL. The default data partitioning method of SIMBA is the same as that of SHadoop. Similarly, as was the case with SHadoop, and as reported in the previous chapter, indexing non-uniformly distributed datasets using the publicly available SIMBA implementation was not possible.

HBase Approaches

Several HBase [41] approaches have been proposed for kNN query processing, such as [18, 44, 45, 63]. For brevity, only [63] and the methods proposed in the previous chapter are summarised in here.

Arguably, the most known kNN query processing method in HBase is MD-HBase [63]. MD-HBase uses k-d tree and quad tree to quantise the space, and Z-ordering to convert multi-dimensional data elements into one-dimensional row-keys. During query processing, as space-filling curves (such as Z-ordering) do not preserve the data locality well, several false positive rows are retrieved [63], which contain irrelevant data to a kNN query. Such a design has a high query response time.

In the previous chapter, a coordinator-based distributed query processing approach that computes a kNN query in a few (tens of) milliseconds was proposed. The primary focus of the proposed approach is to organise, store, and index data in a manner that allows accesses to few but relevant data elements; accordingly, it achieves up to three orders of magnitude performance gains as compared to SHadoop and SIMBA. In the previous chapter, two solutions were proposed: COWI and CONI. Both the solutions partition the domain space by using a quad tree (QT): QT is a member of disjoint decomposition and space partition methods.

Partition Method.	Pros	Cons
Uniform Grid	1,6	2,3,4
R-Tree with STR	3,5,4	2
SFC	1,6	4,5
Q-Tree	3,4,5,	1, 2, 6

Table 4.1: Pros and cons of mv data partitioning methods in parallel/distributed systems

In QT, when the number of data elements in a quadrant exceeds a user-defined threshold, the quadrant is partitioned into 2^d disjoint equal-sized sub-quadrants (cells), where d is the number of dimensions. In COWI/CONI, the QT is constructed on the basis of a small sample drawn uniformly from the input data, and COWI stores the QT in memory, whereas CONI stores the index in a key-value data store (HBase).

The skewness (load imbalance) of the data partitioning process of the space index methods, such as QT and uniform grid, is hardly affected by the size of the sample data [34]. Uniform grid does not depend on a sample to partition a dataset and has a small index building time. The load balance of QT is not affected by the sample [34], but QT has a high index building time [73].

Evaluation Of Multi-dimensional Indexing Approaches in Big Data Systems

In a recent survey [73], MapReduce partitioning methods were evaluated on the basis of several criteria, including (1) index storage space, (2) index building time for non-uniform data, (3) query performance for non-uniform data, (4) number of data points accessed per query (data transfer over the network) for non-uniform data, (5) data proximity preservation, (6) applicability in high dimensions, etc. Recall that tree-based indexing approaches are suitable only in a low-dimensional data space; when such approaches are used, a linear increase in dimension has an exponential increase in the query response time and the memory space requirements [47]. Thus, in this context, high dimensions implies low-dimensional data, mostly less than 10-15 dimensions; for further information, please consult [47].

The findings of the survey [73] are summarised in Table 4.1. The survey concluded that QT provides better data proximity, efficiency for search queries, and low network transfer overhead as compared to other data partitioning methods. However, QT requires high index storage space and index building time and has poor applicability in high dimensions.

In a similar vein, the authors of SHadoop in another work [34] ran extensive experiments in MapReduce to measure the index building time of multi-dimensional index approaches, such as uniform grid, R-tree, R*-tree[14], QT, KD-tree, Z curve, and Hilbert curve. Their main conclusion is that the index building times of all these methods are more or less the

same as the indexing process is dominated by the MR job that scans the entire dataset. The main factor that affects the index building time is the *in-memory step*, which operates on a small sample, and the authors recommend that this opens the space to incorporate more complicated techniques.

Note that in addition to SHadoop, COWI/CONI, and SIMBA, the recent Spark-based methods [76, 86]) also adopt similar indexing techniques. However, even though nowadays we have a large aggregate memory in modern clusters, all the state-of-the-art methods store their global index in a single machine. For instance, the global index of SIMBA is stored in a machine that contains the driver class, that of COWI is stored in the coordinator, and that of SHadoop is stored in the master node. Storing the index in one machine can work well for small and medium-sized datasets, but for a very large-scale dataset, loading a big index in one machine is not desired, as it violates several tenets of big data such as fast to recover(as the size of the index can be massive), easy to back up, resilient, and robust.

Last but not the least, the main conclusion is that the current scalable kNN query processing strategies suffer from significant drawbacks, particularly for non-uniformly distributed datasets. In particular, all the related works (1) fail to address the obvious efficiency and scalability problems that result from the increasing index sizes, (2) struggle to index non-uniform big data as the right size of a sample has to be determined to avoid issues with load balance; or (3) face problems related to either poor data proximity preservation or poor applicability in higher dimensions. Fundamentally, keeping in mind that datasets can be massive and that any system would be called upon to execute a large number of different query types (not just kNN queries), the luxury of kNN query processing using large indexes that grow with the size of a dataset is simply not affordable. Further, keeping the index on distributed storage systems such as HBase simply introduces additional considerable disk and network IO costs into the query processing critical path. This chapter puts forth a new perspective toward kNN query efficiency and scalability, while using negligible space to store an index in memory, the proposed method can easily index non-uniform data or provide a method to measure how good a sample is before trying to index the big data, preserve data locality, and provide high query processing efficiency in high ($\not>$ 10-15) dimensions.

4.4 Data Space Transformation Organization

Thus far, it has been explained that the simple uniform grid-based method performs exceptionally well over datasets that have a uniform distribution. This is because of the following facts: (i) as all cells are equal in size, it enjoys good storage load balance; (ii) as all cells have equal width, all that is required to identify a cell to which a data element belongs, is the width of the cell (i.e. no memory-hungry index is needed); (iii) as its memory footprint

is small and does not increase with the size of the dataset, the domain space can be divided into small cells (this ensures high query performance and scalability); and (iv) it has good applicability in higher dimensions. However, if the distribution of a dataset is non-uniform, as shown in Table 4.1, the uniform-grid indexing method has poor index building time, query performance, and space utilisation.

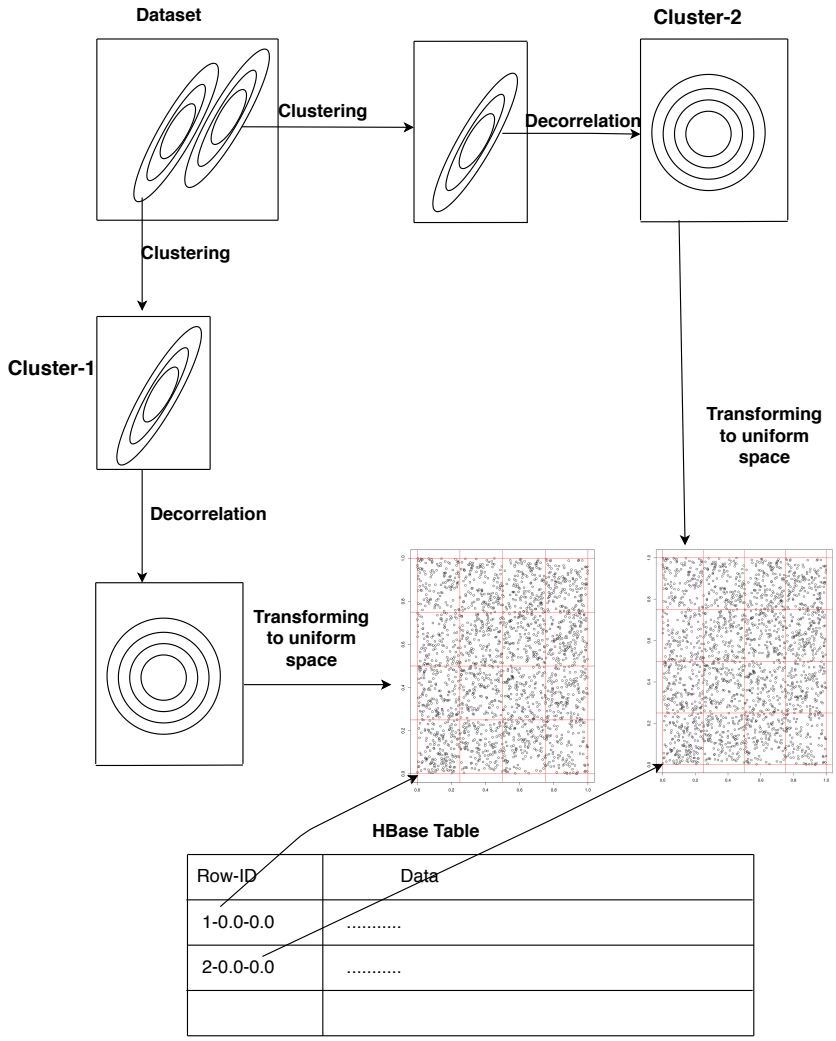


Figure 4.1: Data Transformation Overview

Fortunately, an arbitrary distribution of a dataset can be transformed into a joint uniform distribution on the basis of well-defined statistical principles. The data transformation process is carried out in multiple steps. For instance, to give an overview of the process, consider

a dataset that has an unknown distribution, and the distribution of the dataset is depicted in the upper-left rectangle of fig. 4.1. The first step to transform the unknown distribution of the dataset into a joint uniform distribution is to cluster the dataset using a Gaussian mixture model (GMM), GMM can approximate the distribution of each cluster to a multivariate Gaussian distribution when some specific criteria are met; this will be explained in detail later. Next, as shown in fig. 4.1, the statistical dependency between random variables (RVs) of each cluster is removed by a process known as decorrelation; again, a brief explanation will be provided later in the chapter. *It is worth to note that after the clustering process, each cluster is considered as a separate entity and the decorrelation processes is carried out for each cluster independent of the other clusters, as shown in fig. 4.1* . Next, the distribution of the independent random RVs of each cluster is transformed into a joint uniform distribution separately. Last but not the least, the uniformly distributed dataset is indexed using the simple uniform-grid indexing method, and the lower-left coordinates of the cells of the uniform-grid are used as the row-keys in the HBase table, as shown in fig. 4.1.

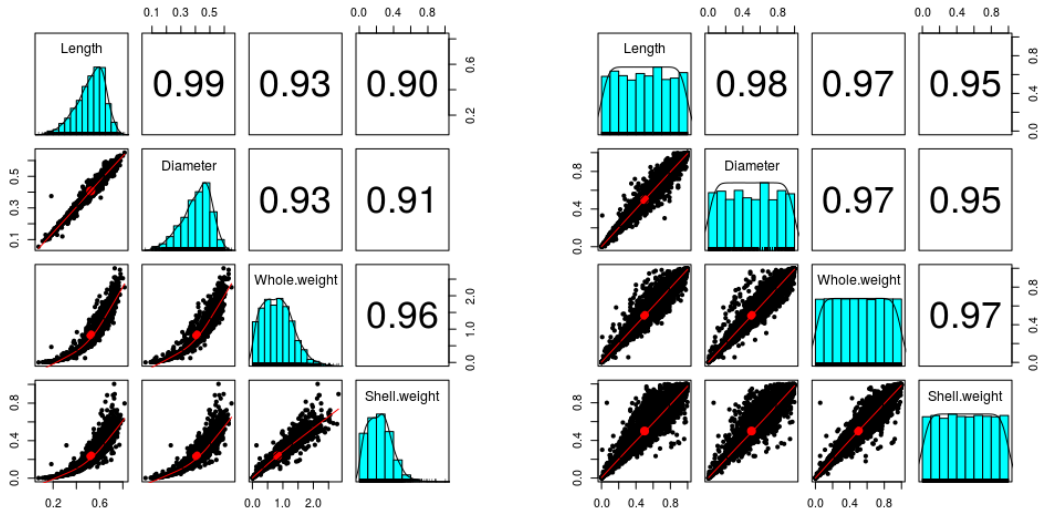
In the remainder of this section, the principles for transforming an arbitrary mv distribution into an mv joint uniform distribution will be explained in detail; these principles are the core ideas behind the new proposed approach.

4.4.1 Statistical Copulas

Statistical probability distribution functions that capture the dependencies between random variables (RVs) of a dataset are called copulas. A marginal distribution, the distribution of each RV, of a copula is the standard uniform distribution, but the joint distribution of a copula is not necessarily uniform; this is a distinctive feature of copulas. According to a well-known statistical theorem, Sklar's theorem (see Theorem 1), any multivariate (mv) distribution can be written in terms of a univariate uniform distribution of each RV and a copula that captures the dependence between the RVs.

Theorem 1. *(Sklar's theorem [74]). Let H be a d-dim. cumulative distribution function $H(x_1, x_2, \ldots, x_d) = P[X_1 \leq x_1, X_2 \leq x_2, \ldots, X_d \leq x_d]$ of random variables $(X_1, X_2, \ldots, X_d)$ with marginals $F_i(x) = P[X_i \leq x_i]$. Then, there exists a copula distribution C with uniform marginals such that $H(x_1, x_2, \ldots, x_d) = C(F_1(x_1), F_2(x_2), \ldots, F_d(x_d))$.*

Example 1: Consider a real dataset called The Population Biology of Abalone – Haliotis species – in Tasmania [83] (hereinafter referred to as Abalone). Abalone has eight RVs; however, for simplicity and to save space, four RVs of Abalone are depicted in a scatterplot matrix in Figure $4.2(a)$; in the scatterplot matrix, each entry of the lower off-diagonal part of the matrix shows the bivariate joint distribution of the corresponding two RVs, the upper off-diagonal entries show the Pearson correlation between the corresponding two RVs, and the



(a) Original Distribution

(b) Copula

Figure 4.2: Original and Copula based Distributions

diagonal entries depict the marginal distribution of each RV. According to Sklar's theorem, these four RVs of Abalone can be expressed by a copula, as shown in Figure $4.2(b)$. Note that the marginal distribution of each RV of the copula, shown in the diagonal entries of the scatterplot matrix of Figure $4.2(b)$, is the standard uniform distribution. Moreover, the pairwise dependencies between any two RVs of Abalone are captured by the mv distribution - copula - (see the lower off-diagonal entries). Note that (i) Pearsons correlations between the original RVs (see Figure $4.2(a)$) and those between the RVs of Figure $4.2(b)$ are relatively same; and (ii) even though each RV in Figure $4.2(b)$ has a uniform distribution, the bivariate distribution between any two of these RVs is not necessarily uniform.

In the literature, several different types of copulas are defined and used for different purposes. In this work, however, only the independence copula is of interest. In the independence copula, the marginal and joint distributions of RVs are the standard uniform, as there is no statistical dependency between the RVs of the independent copula. If a dataset can be defined by the independence copula, on the basis of the joint uniform distribution of the copula, the dataset can be indexed using the simple uniform-grid index. Therefore, the memory-hungry tree-based multi-dimensional indexes are not required.

However, as the RVs of most of the real-world datasets are not statistically independent of each other, an independent copula has limited applications in real-world datasets. Recall that independent RVs have zero correlation, but the inverse is not always true; thus, removing the correlation between the dependent RVs does not yield independent RVs. However, in a special case, when the marginal and joint distributions of the RVs of a dataset belong to elliptical distributions, removing the correlation between RVs implies independence [67].

However, yet again, not all the real-world datasets are generated by elliptical distributions.

According to [58], any mv continuous distribution can be approximated arbitrarily well by finite GMMs to arbitrary accuracy [58]; as a Gaussian distribution is an elliptical distribution, removing the correlation of the RVs of a component (cluster) of the GMM implies independence.

In summary, any arbitrary distribution of a dataset can be transformed into a joint uniform distribution by using the following steps:

- clustering: approximate the distribution of the dataset by using GMM.
- apply the following steps for each cluster separately:
  - decorrelating: remove the statistical correlation among the RVs.
  - marginal uniform: transform the marginal distribution of every RV to the standard uniform.
  - testing the goodness of fit: check whether every pair of RVs can be defined by an independent copula.

These steps are explained in detail next.

4.4.2 Data Clustering Methodology

Gaussian mixture models (GMMs) are used for clustering data points that are heterogeneous and stem from different sources. A GMM models the density of mv RVs as a weighted sum of the Gaussian density and is defined as follows:

$$f(x) = \Sigma_{m=1}^{M} \pi_m \psi_m(\mathbf{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m) \tag{4.1}$$

where $\mathbf{x}$ is an RV, $\psi_m(\mathbf{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ is the Gaussian density with mean vector $\boldsymbol{\mu}_m$ and covariance matrix $\boldsymbol{\Sigma}_m$, and π_m are the positive mixing weights that satisfy the constraint $\Sigma_{m_1}^{M} \pi_m = 1$. Given that M is the smallest integer such that $\pi_m > 0$ for $1 \leq m \leq M$, and $(\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \neq (\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$ for $1 \leq a \neq b \leq M$, the complete set of parameters of GMM $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_M, \pi_1, \ldots, \pi_M\}$ can be estimated by using the maximum likelihood method via the EM algorithm[32].

A GMM uses an estimated number of clusters when the number of clusters in a dataset is not known in advance. If the estimated number of clusters is too small, the GMM might not be able to approximate the true density of the dataset. If the estimated number of clusters is too high, the GMM might over-fit the data [24]. Estimating the right number of clusters is a

non-trivial problem [46] and has a significant effect on how well a cluster can be transformed into an mv joint uniform distribution.

Arguably, in the literature, the Bayesian information criterion (BIC) [72] is the most popular method for estimating the number of clusters in the GMM. In this work, BIC is used for estimating the number of components, M, of a dataset. Keep in mind that the proposed method does not depend on BIC, and any consistent estimator of the correct number of clusters can be equally adapted.

After the clustering of the sample data with the GMM, the bootstrapping [80] method is used for estimating the statistic M, which is the number of clusters that can be derived over the entire dataset and the clustering coefficients of the GMM. In particular, the bootstrapping methodology estimates the standard errors of all the parameters of the GMM $\{\pi_m\}_{m=1}^{M}$. It creates another sample of the same size as the original sample by re-sampling with the replacement from the original sample. The resulting bootstrapping distribution of the statistics of interest (number of clusters $|M|$ and the corresponding clustering parameters) is then derived, where the required GMM statistics are obtained. After clustering a dataset, one can remove the correlation between the RVs of a cluster to transform the RVs into independent RVs.

4.4.3 Removing Statistical Correlation

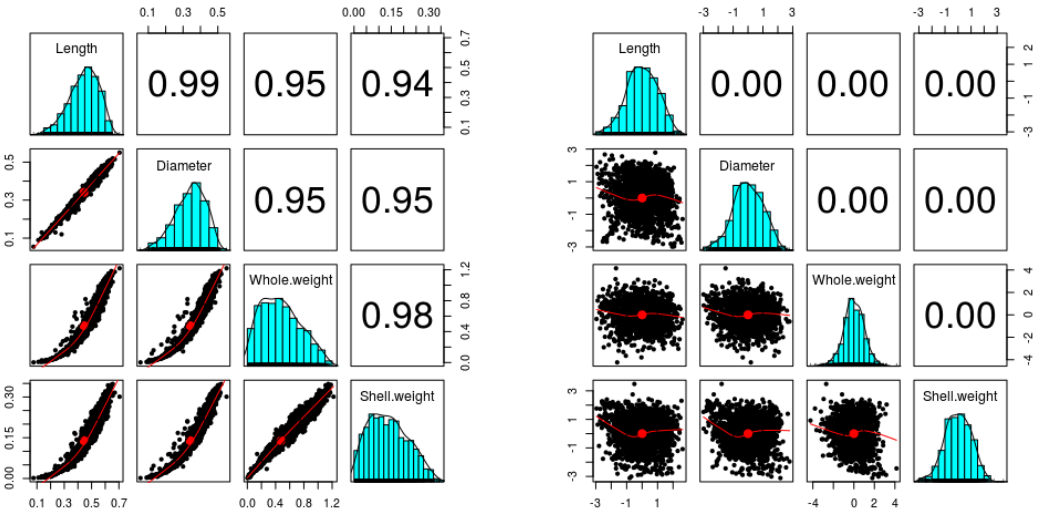
One of the most important features of copulas is flexibility; copulas are flexible enough to separately (i) model the dependence between the RVs and (ii) transform the RVs into marginal uniform distributions. Accordingly, next, the process of removing the dependencies between RVs is explained first; then, the process of transforming each RV into a uniform distribution will be discussed.

After the clustering of a dataset by using the GMM, each cluster is approximated by a Gaussian distribution that is defined by a mean vector ($\boldsymbol{\mu}$) and a co-variance matrix ($\boldsymbol{\Sigma}$). As mentioned earlier, since the Gaussian distribution is a member of an elliptical distribution, removing the correlation between the RVs of a cluster implies transforming the RVs into independent RVs.

Therefore, the correlation between RVs, say of vector $\mathbf{x}$ such that $\mathbf{x} \in \mathbf{M}_i$, where $\mathbf{M}_i$ is the ith cluster of a GMM, can be removed by multiplying $\mathbf{x}$ by the *whitening* matrix $\mathbf{A}$:

$$\boldsymbol{\Sigma}^{-1} = \mathbf{A}^T\mathbf{A}, \tag{4.2}$$

where $\boldsymbol{\Sigma}^{-1}$ is the inverse of the covariance matrix of $\mathrm{M_i}$. Usually, the Cholesky decomposition is used to estimate matrix $\mathbf{A}$ from $\boldsymbol{\Sigma}^{-1}$. Hence, a transformed vector $\boldsymbol{x}'$, whose RVs are



(a) Original Data Space

(b) Independent Data Space

Figure 4.3: Transforming To Independent Data Space

statistically independent of each other, is defined as follows:

$$\boldsymbol{x}' = \mathbf{A}_i(\boldsymbol{x} - \boldsymbol{\mu_i}) \tag{4.3}$$

where $\mathbf{A}_i$ and $\boldsymbol{\mu}_i$ are the whitening matrix and the mean vector of $\mathbf{M}_i$ of the ith cluster, respectively.

Example 2: Using the GMM, after clustering the four RVs of the Abalone dataset into two components, we obtain the scatter plot matrix of one of the components, as shown in Figure $4.3(a)$. Between the RVs of the cluster, as shown at the upper off-diagonal of Figure $4.3(a)$, strong correlations exist; however, applying equation 4.3 to each data element that belongs to the cluster removes the correlations between the RVs as shown at the upper off-diagonal plots of Figure $4.3(b)$. Thus, this implies that the original RVs of the cluster (see the lower off-diagonal of Figure $4.3(a)$) are transformed into independent RVs (see the lower off-diagonal of Figure $4.3(b)$).

4.4.4 Transforming to Uniform Data Space

The probability integral transformation (PIT), a well-known statistical theorem, see Theorem 2, states that the *cdf* of an RV has the standard uniform distribution. In this work, therefore, PIT is applied to transform the marginal distribution of the independent RVs into the standard uniform distribution.

Theorem 2. *(Probability Integral Transformation [22, chapter 2, p. 54]). Let random variable Y have a continuous distribution with cumulative distribution function (cdf) $F_Y(y) =$*

$P(Y \leq y)$ and define the random variable $U = F_Y(y)$. Then, U is uniformly distributed in (0,1) with $P(U \leq u) = u, 0 < u < 1$.

Proof. ([22, chapter 2, p. 54])

$$\begin{aligned} P(U \leq u) &= P(F_Y \leq u) = P(F_Y^{-1}[F_Y] \leq F_y^{-1}(u)) \\ &= P(Y \leq F_y^{-1}(u)) = F_y(F_y^{-1}(u)) = u \end{aligned} \tag{4.4}$$

□

By convention, the RV Y generated by a continuous probability distribution function (pdf) is denoted by $p(y)$ and the cumulative distribution function of Y is denoted by $F_Y(y)$. The relationship between $p(y)$ and $F_Y(y)$ of the RV Y is as follows: if $\int_{-\infty}^{+\infty} p(y)dy = 1$, then there exists another continuous random variable $u = F_Y(y) = \int_{-\infty}^{y} p(y)\, dy = P(Y \leq y)$; thus, u is called a cdf of y, and as such, U is a monotonic non-decreasing function of Y, where $0 < u < 1$. To compute the cdf of the RV Y, we need to solve $F_Y(y) = \int_{-\infty}^{y} p(y)\, dy$, but for many distributions, the integral is not available in a closed form. Hence, the cdf of such RVs can be computed empirically:

$$\hat{F}_Y(y) = \frac{1}{n}\Sigma_{i=1}^{n} \mathbb{1}_{x_i \leq y} \tag{4.5}$$

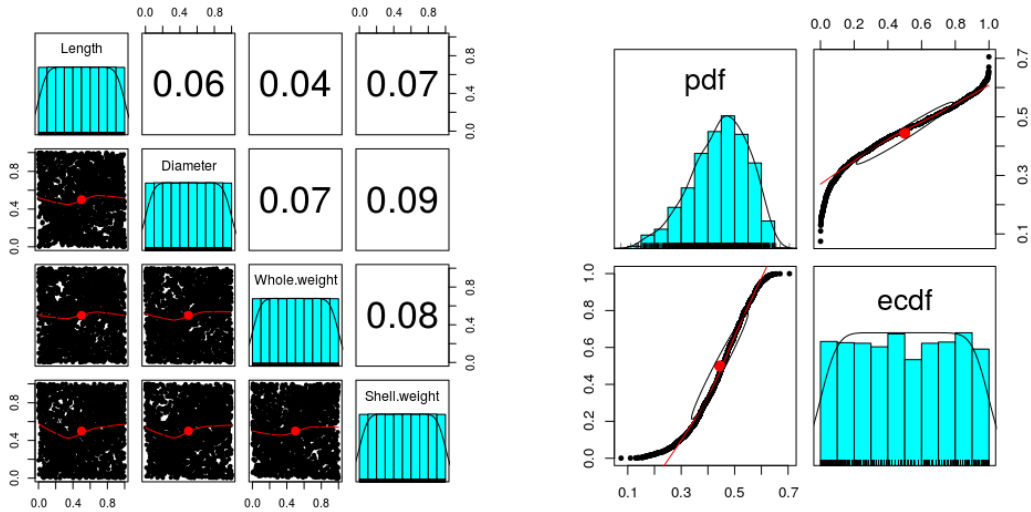
where x_i is the ith data point in the dataset, n is the total number of data points in the dataset, and $\mathbb{1}_{x_i} \leq y$ = 1 if $x_i \leq y$, otherwise $\mathbb{1}_{x_i} \leq y = 0$.

Example 3: Consider one of the RVs of Abalone called Length. To compute the $ecdf$ of the RV, all the n values in the RV must be sorted in the ascending order. Then, starting from zero, and by jumping $1/n$ for each of the n data points, a monotonic increasing function is drawn between 0 and 1; see the upper right and the lower left of Figure 4.4(b). The upper-left histogram of Figure 4.4(b) shows the marginal pdf of Length; whereas the lower-right histogram shows the marginal cdf of Length, and hence, as stated in the PIT theorem, the cdf of the RV, Length, has the standard uniform distribution.

For the sake of completeness, (4.5) is provided to express the cdf of an RV as a uniform distribution. However, computing cdf in such a manner is inefficient, particularly when dealing with a large-scale dataset; thus, the cdf of the standard normal distribution is computed as follows:

$$F(x) = \int_{-\infty}^{x} \frac{\exp^{-x^2/2}}{\sqrt{2\pi}}, \tag{4.6}$$

where the integral is not available in a closed form and is approximated numerically [27].



(a) Uniform Data Space

(b) Length RV PDF and ECDF

Figure 4.4: The Uniform Space and CDF

4.4.5 Goodness of Fit

Thus far, all the essential building blocks for transforming an arbitrary distribution to a joint uniform distribution are explained, and pictorial illustrations are provided along every step. For instance, Figure $4.4(a)$ depicts a pair-wise joint uniform distribution between the four RVs of abalone. A visual examination of the pair-wise distributions revealed that the RVs were fairly transformed into a joint uniform distribution; this can be used as the first step of evaluating how well RVs are transformed into the uniform distribution.

However, a statistical method of testing the uniformity of the transformed RVs is crucial. In this work, a well-known statistical method that is used for testing a pair-wise independence copula is used; according to [40], a pair-wise dependency between two RVs of the independent copula was determined using Kendall's Tau (denoted by τ). When two random variables Y_1 and Y_2 are independent, the distribution of τ is close to a normal distribution with zero mean and variance $\frac{2(2n+5)}{9n(n-1)}$ [40]. Thus, the p-value for dependency test is computed as:

$$\begin{aligned} \text{p-value} &= 2(1-\phi(T)), \\ T &= \sqrt{\frac{(9n(n-1))}{(2(2n+5))}} \times |\tau| \end{aligned} \tag{4.7}$$

When two random variables Y_1 and Y_2 are independent, the distribution of τ is close to a normal distribution with zero mean and variance $\frac{2(2n+5)}{9n(n-1)}$ [40]. Thus, the p-value for the

Table 4.2: p-values of bi-variate Independence copula tests

	Length	Diameter	W.weight	S.weight
Length	-	0.4132669	0.7220138	0.5368411
Diameter	0.4132669	-	0.6690015	0.5266671
W.weight	0.7220138	0.6690015	-	0.3699861
S.weight	0.5368411	0.5266671	0.3699861	-

dependency test is computed as follows:

$$\begin{aligned} &\text{p-value} = 2(1-\phi(T)), \\ &T = \sqrt{\frac{(9n(n-1))}{(2(2n+5))}} \times |\tau| \end{aligned} \tag{4.8}$$

where ϕ is the standard normal distribution. Therefore, one can accept the null hypothesis (i.e. the two RVs are independent) at 95% level of acceptance when the p-value is ≤ 0.05.

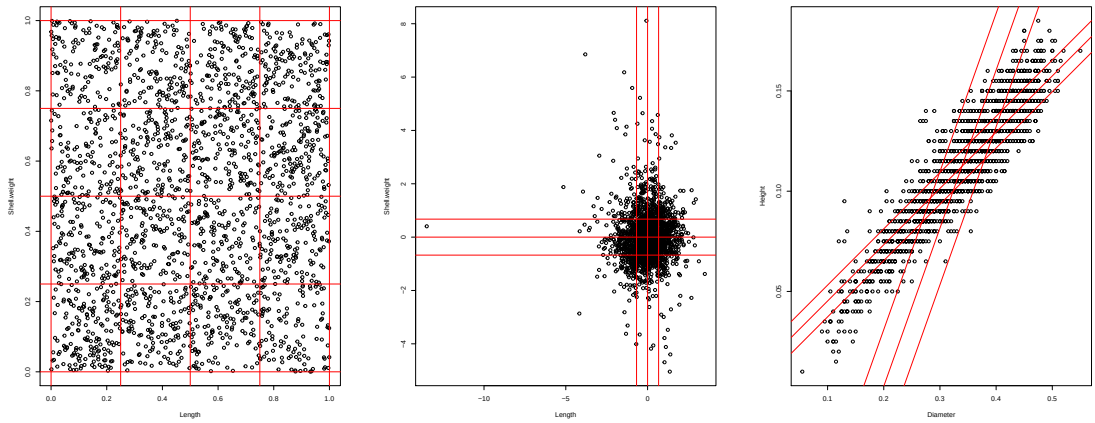
For example, pairwise dependency tests for the four RVs of Abalone were carried out, and the p-value results are provided in Table 4.2; as none of the p-value results was less than 0.05, it was safe to accept the null hypothesis; i.e. the original distribution of Abalone (Figure $4.3(a)$) was successfully transformed into an mv standard uniform distribution (Figure $4.4(a)$).

4.5 Data Space Transformation Approach

4.5.1 Comparison with Tree-Based Approaches

Intuitively, a dataset that has a joint uniform distribution can be easily partitioned into $|\mathcal{C}|$ equal-sized cells. For a visual illustration in the 2-d space, consider two RVs, namely the Length and the Shell Weight from the dataset Abalone. After transforming the distribution of the two RVs into a joint uniform distribution, we can partition the uniform domain space into $|\mathcal{C}| = N/\alpha$ equal-sized cells (see Figure 4.5(a)). Each data partition (cell) contains on average α data elements, where N is the total number of data elements in the dataset.

The uniform domain space can be an inverse-transform to the independent space, (Figure 4.5(b)), and the independent domain space can also be an inverse-transform to the original domain space, (Figure 4.5(c)). During the inverse-transformation process, the size of a cell in the independent and/or original space might shrink, expand, and/or rotate; therefore, only cells in the uniform space are of equal size. In contrast to the cell size, all the corresponding



(a) Uniform space (b) Indep. space (c) Original space

Figure 4.5: Data Domain Spaces

cells of the three domain spaces contain the same number of data elements (as will be proven shortly).

Recall that disjoint tree-based index approaches divide a data space into several cells that might be inequal in size but contain an approximately equal number of data elements. Similarly, the proposed approach partitioned the non-uniform domain spaces, (Figure 4.5(b) and 4.5(c)), into several inequal cells that contain an equal number of data elements. Hence, the proposed method can partition a domain space into small cells, in a similar manner as the tree-based approaches do, without building and maintaining a memory-hungry tree-based data structure. This demonstrates that the proposed method has a significant advantage in terms of scalability and memory requirements.

4.5.2 Distance in The Three Domain Spaces

Intuitively, during the aforementioned transformation process, the domain spaces can be rotated, shrunk, or stretched. Hence, the distance between any two data elements in the original space can be different from the distance between the corresponding data elements in the independent space (and similarly in the uniform space). Consequently, given a data element $\mathbf{q}$ in the original space and its projections $\mathbf{q}^{ind}$ and $\mathbf{q}^{uni}$ in the independent and uniform spaces, respectively, the kNN data elements to $\mathbf{q}$ in the original space do not necessarily map to the kNN data elements to $\mathbf{q}^{ind}$ in the independent space or to the kNN data elements to $\mathbf{q}^{uni}$ in the uniform space. However, as proven in Lemma 2, the Euclidean distance between the data elements in the independent space and the Mahalanobis distance between the corresponding data elements in the original space are the same.

Lemma 2. *Euclidean distance between data points in the independence space is the same as Mahalanobis distance between data points in the original space.*

Proof.

$$
\begin{aligned}
d_M^2(\mathbf{x}^{ind}, \mathbf{y}^{ind}) &= (\mathbf{x}-\mathbf{y})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\mathbf{y}) \\
&= (\mathbf{x}-\mathbf{y})^T \boldsymbol{A}^T \boldsymbol{A} (\mathbf{x}-\mathbf{y}) \\
&= (\mathbf{x}-\mathbf{y})^T \boldsymbol{A}\boldsymbol{A} (\mathbf{x}-\mathbf{y}) \\
&= [\boldsymbol{A}(\mathbf{x}-\mathbf{y})]^T [\boldsymbol{A}(\mathbf{x}-\mathbf{y})] \\
&= [\boldsymbol{A}\mathbf{x} - \boldsymbol{A}\mathbf{y}]^T [\boldsymbol{A}\mathbf{x} - \boldsymbol{A}\mathbf{y}] \\
&= [\mathbf{x}^{ind} - \mathbf{y}^{ind}]^T [\mathbf{x}^{ind} - \mathbf{y}^{ind}] \\
&= \Sigma_{i=1}^{d} (x_i^{ind} - y_i^{ind})^2
\end{aligned}
$$

N.B as $\boldsymbol{\Sigma}$ is a square matrix $\boldsymbol{A}^T = \boldsymbol{A}$. □ □

Similarly, the distance between the data elements of the uniform space and the distance between the corresponding data elements of the independent (or original) space is not the same. Hence, computing kNN with the data elements of the uniform space does not necessarily produce the correct results. For example, without any loss of generality, consider in a 1-d space, a fictitious sorted tuple $\mathbf{p} = \{1.23, 2.2, 5.6, 70.0, 80.9\}$, and *cdf* of each value of $\mathbf{p}$, computed empirically using eq. (4.5), is stored in $\mathbf{p}^u$; hence, $\mathbf{p}^u = \{0.2, 0.4, 0.6, 0.8, 1.0\}$. If one wants to compute the three nearest neighbours (3-NN) of a number, say 5.6, on the basis of the uniform space; first, the number 5.6 has to be transformed into the uniform space, as the *cdf* of $x \leq 5.6$ is 0.6. Then, the 3-NN to 0.6 over the uniform space (tuple $\mathbf{p}^u$) are 0.6, 0.4, and 0.8, respectively. However, when these values are mapped back to the original data space (of tuple $\mathbf{p}$), the corresponding values are 5.6, 2.2, and 70.0, respectively. However, the 3-NN to 5.6 computed on the basis of the original space are 5.6, 2.2, and 1.23, respectively; this demonstrates that kNN computed on the basis of the uniform space are not the same as the kNN computed on the basis of the original space. Thus, computing a kNN query in the uniform space does not necessarily produce correct results.

However, when all the data elements contained in a cell of the uniform space are transformed back to the independent space, all the transformed data elements reside within a cell of the independent space that corresponds to the cell of the uniform space. In the same vein, if data elements that reside in a cell of the independent space are transformed back to the original space, all the transformed data elements in the original space reside in a cell of the original space that corresponds to the cell of the independent space. Thus, the corresponding cells in the three domain spaces contain precisely the same data elements, but the data elements are defined according to the domain space to which a cell belongs. This will be elaborated in detail in the following sub-section.

4.5.3 Data Preservation in the Corresponding Cells

Although the distance between data elements is not the same in the three domain spaces, the data locality, i.e. data elements contained in a cell, is preserved across the three domain spaces. To back this claim up, the hypothesis that for all the data elements that lie within a cell in the uniform space, all of the corresponding data elements also lie within the corresponding cells in the independent and in the original data spaces has to be proven; this proof is provided in Lemma 3.

Lemma 3. *For all data points that reside within a cell in the uniform space, their corresponding data points lie within the boundaries of the corresponding cell in the independent and original spaces.*

Proof. Let u_i be the value of the ith dimension of a data point $\mathbf{u}$ that lies in cell C^{uni} in the uniform space such that $[c_i^{uni\_min} \leq u_i \leq c_i^{uni\_max}]$, where $c_i^{uni\_min}$ and $c_i^{uni\_max}$ are the minimum and the maximum values of the ith dimension of C^{uni}, respectively. Let data point $\mathbf{y}$ be the point resulting from transforming $\mathbf{u}$ to the independent space with y_i being its value in the ith dimension and cell C^{ind} resulting from transforming C^{uni} to the independent space with $c_i^{ind\_min}$ and $c_i^{ind\_max}$ being the minimum and the maximum values of the ith dimension of c^{ind}, respectively. Similarly, consider a cell C^{org} and a data point $\mathbf{x}$ created by transforming C^{ind} and $\mathbf{y}$ to the original space, respectively. If it is proven that a point belongs to C^{ind} iff it belongs to C^{uni}, then, on the basis of transitivity, it is sufficient to show that a point belongs to C^{org} iff it belongs to C^{ind}.

Case 1: Independent and Uniform Data Spaces. One needs to prove that $[c_i^{ind\_min} \leq y_i \leq c_i^{ind\_max}]$ is true. Each RV of the uniform space is a *cdf* of an RV in the independent space:

$$\begin{aligned}
&c_i^{uni_min} \leq u_i \leq c_i^{uni_max} \Leftrightarrow \\
&F_Y(Y \leq c_i^{ind_min}) \leq F_Y(Y \leq y_i) \leq F_Y(Y \leq c_i^{ind_max}).
\end{aligned}$$

Since *cdf* is a monotonic increasing function and let ϕ be the inverse of *cdf*,thus:

$$\begin{aligned}
&\phi(c_i^{uni_min}) \leq \phi(u_i) \leq \phi(c_i^{uni_max}) \\
&\Leftrightarrow F_Y^{-1}(Y \leq c_i^{ind_min}) \leq F_Y^{-1}(Y \leq y_i) \leq F_Y^{-1}(Y \leq c_i^{ind_max}) \\
&\Leftrightarrow c_i^{ind_min} \leq y_i \leq c_i^{ind_max}.
\end{aligned}$$

Case 2: Independent and Original Data spaces. Now, it has been proven that data point $\mathbf{x}$ lies within C^{org} iff $\mathbf{y}$ lies within C^{ind}. Without any loss of generality, consider that the data points $\mathbf{c^{org\_min}}$, $\mathbf{x}$, and $\mathbf{c^{org\_max}}$ are perpendicular and $\mathbf{c^{org\_min}}$ and $\mathbf{c^{org\_max}}$ are located on the boundaries of C^{org}. Consider that $\mathbf{c^{ind\_min}}$ and $\mathbf{c^{ind\_max}}$ are the corresponding points to $\mathbf{c^{org\_min}}$ and $\mathbf{c^{org\_max}}$, respectively, and are located on the boundaries of C^{ind}. To prove

by contradiction, let us assume that $\mathbf{x}$ does not lie within C^{org} when $\mathbf{y}$ lies within C^{ind}. As $\mathbf{A}^{-1}$ is the inverse of the whitening matrix in eq. (4.2), recall that the Gaussian components, clusters, are transformed independent of each other. Thus, each component has its own transformation spaces (original, independent, and uniform spaces); moreover, each cluster has its own whitening matrix and $\mathbf{A}^{-1}$ matrix. Note that transformation is carried out for each cluster separately; hence:

$$(c_i^{org\_min} > x_i) \vee (x_i > c_i^{org\_max}) \Leftrightarrow$$
$$((\mathbf{A}^{-1}\mathbf{c}^{ind\_min})_{i0} > (\mathbf{A}^{-1}\mathbf{y})_{i0}) \vee ((\mathbf{A}^{-1}\mathbf{y})_{i0} > (\mathbf{A}^{-1}\mathbf{c}^{ind\_max})_{i0})$$

By multiplying both sides by $\mathbf{A}$, one can obtain:

$$\begin{aligned}
&((\mathbf{A}\mathbf{A}^{-1}\mathbf{c}^{ind_min})_{i0} > (\mathbf{A}\mathbf{A}^{-1}\mathbf{y})_{i0}) \\
&\vee ((\mathbf{A}\mathbf{A}^{-1}\mathbf{y})_{i0} > (\mathbf{A}\mathbf{A}^{-1}\mathbf{c}^{ind_max})_{i0}) \\
&\Leftrightarrow ((\mathbf{I}\mathbf{c}^{ind_min})_{i0} > (\mathbf{I}\mathbf{y})_{i0}) \vee ((\mathbf{I}\mathbf{y})_{i0} > (\mathbf{I}^{-1}\mathbf{c}^{ind_max})_{i0}) \\
&\Leftrightarrow (c_i^{ind_min} > y_i) \vee (y_i > c_i^{ind_max}).
\end{aligned}$$

The last equivalence does not hold when $\mathbf{y}$ lies within C^{ind}; hence, our assumption that $\mathbf{x}$ does not lie within C^{org} when $\mathbf{y}$ lies within C^{ind} must be wrong. That is, $\mathbf{x}$ lies within C^{org} iff $\mathbf{y}$ lies within C^{ind}. □

Data locality is an important building block for computing the exact kNN of the proposed approach. Armed with such knowledge, the proposed method can compute the exact kNN, and not approximated kNN, as explained next.

4.5.4 Computation of Exact kNN

Computing the exact kNN might require accessing several neighbouring cells, but in the original (or independent) domain space, the identification of all the neighbouring cells is impossible without building memory-hungry indexes. Fortunately, in the uniform space, building a simple uniform-grid index is sufficient to retrieve all the neighbouring cells that are required for computing the exact kNN query. For instance, assume that cells are represented by their lower-left coordinates and a cell, in which a query lies, is identified using a simple arithmetic operation, as shown in the example in section 3.7.2. The lower-left coordinates of a neighbouring cell to the cell that contains the query can be computed by adding or subtracting r, the width of the grid-cells, to or from the ith dimension of the lower-left coordinates of the cell that contains the query.

Computing the exact kNN on the basis of the transformation spaces depends on the following three facts. (F1) A simple uniform grid can efficiently index the uniform space only. (F2)

Only the data elements of the original domain space are needed for computing kNN. (F3) The corresponding cells in the three domain spaces contain the same data elements that are represented differently according to their domain space (data locality). On the basis of these facts, a new method of data organising and indexing is proposed for computing the exact kNN.

Accordingly, after a dataset is transformed and the uniform space is partitioned into equal-sized cells, each cell of the uniform domain space instead of storing the data elements of the uniform domain space stores the corresponding data elements of the original domain space. Such a design philosophy has two distinct features: (i) all the neighbouring cells can be identified easily, without having the memory-hungry indexes, as the cell of the uniform space has equal width; and (ii) the exact kNN can be computed correctly, as the data points of the original domain space are stored in the uniform cells.

At the first glance, such a design, a hybrid of the uniform and original domain spaces, might seem counter-intuitive, because usually only the original domain space is used for indexing a dataset, for example as in the case of tree-based index approaches. However, the lower-left coordinate of a cell of the original domain space can be defined as a function of the lower-left coordinate of the corresponding cell of the uniform space as follows:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{K}(\mathbf{u}), \tag{4.9}$$

where $\mathbf{x}$ is the lower-left coordinate of a cell of the original space, $\mathbf{A}^{-1}$ is the inverse of the whitening matrix, K is a function (inverse cdf of RVs) that transforms back the RVs of the uniform space to the RVs of the independent space, and $\mathbf{u}$ is the lower-left coordinate of the corresponding cell of the uniform space. Thus, this can be seen as representing the *inequal-sized cells* of the original space by the *equal-sized cells* of the uniform space.

4.6 Data and Query Processing

This section describes the technical details of the proposed solution named Space Transformation Organisation Structure (STOS). First, the creation of the STOS will be discussed, then kNN query processing in STOS will be explained.

4.6.1 Creation of the STOS

The creation of the STOS can be broken down into two phases: data pre-processing and data organisation.

The **Data Pre-processing Phase** proceeds as follows:

1. Using a MapReduce job, draw a uniform random sample from a dataset and count the total number of data elements in the dataset.
2. Operating on the sample, determine the correct number of GMM components using BIC.
3. Use GMM (eq. (4.1)) and bootstrap to cluster the sample, then compute the mean vector, the covariance matrix, and the positive mixing weight of each cluster.
4. For each cluster, determine the whitening matrix **A** (eq. (4.2)) and decorrelate each cluster.
5. For each cluster, estimate cdf of the cluster (eq. (4.6)) and transform the distribution of the decorrelated cluster (independent space) into mv uniform distribution.
6. For each cluster, run pairwise dependence test at 95% level of accuracy; if the test fails either start from step (2) using a different number of clusters (or reduce the level of accuracy).
7. For each cluster, using the positive mixing weights and the total number of data elements in a dataset, compute a total number of data elements that belong to a cluster. Afterwards, based on a user-defined average number of data elements to be stored per cell, using (definition 10), determine the width of cells of a uniform-grid of a cluster.

The **Data Organisation Phase** consists of a MR job where the mappers go through the data elements at their input. For each data element, a Mapper:

1. assigns it to a cluster using a naive Bayesian algorithm;
2. transforms the data element to the uniform space based on the parameters, computed in the previous phase, of the cluster to which the data element belongs to;
3. based on definition 10 and definition 11, assigns the data element to a cell of a uniform-grid of the cluster, in which the data element resides.
4. finally, emit a key-value pair, where the key is a concatenation of the cluster-ID (a random number given to identify a cluster) and lower-left coordinates of the cell, and the value is the actual value of the data element of the original space.

Afterwards, the emitted key-value pairs from the Mappers are grouped and sorted by their key. Then, the reducers simply write out the re-organised data into data files, containing the ordered key-value pairs themselves (key: cluster-id plus lower-left coordinates of a cell; value: a collection of data elements that reside in the cell). The reducers also compute per-cluster metadata (MBRs) that contain, for every dimension of each cluster, the minimum and maximum value. This metadata, together with the parameters computed during the previous phase, constitute the in-memory state of the proposed approach. Please note that the space overhead for this is expected to be minimal. The proposed methods stores, per cluster, the mean vector ($d \times sizeof(double)$), the covariance matrix ($d^2 \times sizeof(double)$), the MBR of the cluster ($2 \times d \times sizeof(double)$) and the cell width ($sizeof(double)$), for a total of $d^2 + 3d + 1$ doubles per cluster, and a grant total of $|\mathcal{M}| \times (d^2 + 3d + 1) \times 8$ bytes across all

$|\mathcal{M}|$ clusters. For example, for spatial – 3-dimensional – data, this would amount to a measly 152 bytes per cluster; most real-world datasets have been seen so far have only a handful of clusters in total!

The reorganised data output by the above reducers have to be stored in HBase to expedite query processing. The HBase table schema follows the aforementioned design; so, each row in this table has a rowkey that is the concatenation of a cluster ID and the lower-left coordinates of a cell of the uniform space, and contains a single column with all data points (in the original space) mapped to that cell.

Inserting the output files from the Reducers into HBase table sequentially, takes considerable time. So, the standard HBase bulk loading [13] technique is used to insert those output files into HBase table.

Please note that one can very easily transform the coordinates of a data element from the uniform to the original space using eq. (4.9). Hence, both the second part of the rowkey and the data points in the columns could be stored in either their original form or their transformed (uniform space) form. But in this work,the later schema is chosen as it is more convenient during the creation of the STOS and query processing.

4.6.2 Query Processing

When a query point, $\mathbf{q}$, arrives then:

1. compute the distance between $\mathbf{q}$ and the MBR of each cluster (definition 4) and the closest cluster is selected.
2. The query point $\mathbf{q}$ is transformed into the uniform space, using the parameters of the selected cluster, resulting in a new point $\mathbf{q}^{uni}$.
3. This new point is mapped to a cell in the cluster using definition 11, and thus the rowkey of the corresponding row in HBase is computed.
4. The contents of the above cell are retrieved from HBase and an initial kNN answer is computed. If $k > \alpha$ (where α is the (average) number of data points per cell – definition 10), then also retrieve as many neighbouring cells as necessary to ensure at least k data elements are fetched.
5. afterwards compute the distance, ρ, between $\mathbf{q}$ and the kth data element of the initial kNN answer, and draw a hyper-square whose centre is $\mathbf{q}$ and whose width is $2 \times \rho$.
6. Last, the rows for all unprocessed cells intersecting and/or covered by the hyper-square are retrieved and the final kNN result is computed and returned.

4.7 Performance Evaluation

The experiments were run on a five-node cluster; each node was a Dell R720 server with four Intel Xeon CPUs (eight cores each), 64-GB RAM, and 2-TB disk space.

Datasets

Two real-world datasets from the UCL machine learning data repository, namely Istanbul stock exchange national 100 index [6] and Activity Recognition system based on Multisensor data fusion (AReM) [65], were used in the experiments. As it is already known that QT has poor applicability in higher dimensions [73], for a fair comparison of STOS with COWI, a QT-based approach, two dimensions were selected from each dataset. From the Istanbul dataset, the first two dimensions were used, whereas from the AReM dataset, the first and the third dimensions (cycling and walking) were adopted. Moreover, to quantify the query performance of the proposed method in the higher dimensions, all the six and the nine dimensions were used from the AReM and Istanbul datasets, respectively.

A publicly available large-scale dataset is difficult to find. Therefore, a prevalent practice in the literature, for example, see [44]), is to generate large-scale datasets based on the parameters of real-world datasets. Consequently, based on the AReM and Istanbul datasets, six datasets of different sizes were generated. The first three datasets were generated on the basis of AReM and contained 8, 16, and 32 billion points, which corresponded to a size of approximately 250 GB, 500 GB, and 1 TB, respectively; simultaneously, the second three datasets were generated on the basis of the Istanbul dataset and contained 8, 16, and 32 billion points, which corresponded to a size of approximately 250 GB, 500 GB, and 1 TB, respectively. Given the standard replication factors in the NoSQL and HDFS, these datasets reached the near-maximum available storage space of the five-node cluster, in which the experiments were carried out. Moreover, for the higher-dimensional data, an additional six datasets were generated. The first three of which were generated on the basis of AReM and contained 2.6, 5.33, and 10.6 billion six-dimensional data and corresponded to the size of approximately 250 GB, 500 GB, and 1 TB, respectively; similarly, based on the Istanbul dataset, the remaining three datasets were generated, and these datasets were composed of 1.7, 3.5, and 7.1 billion nine-dimensional data that corresponded to the size of approximately 250 GB, 500 GB, and 1 TB, respectively.

Queries and Performance Metrics

Queries. 10k queries per dataset were generated on the basis of the distribution of the datasets; the queries were used for the performance evaluation. For each query, the system computed kNN over a dataset, based on which the queries were generated. This was repeated for each value of $k \in \{10, 100, 1000\}$.

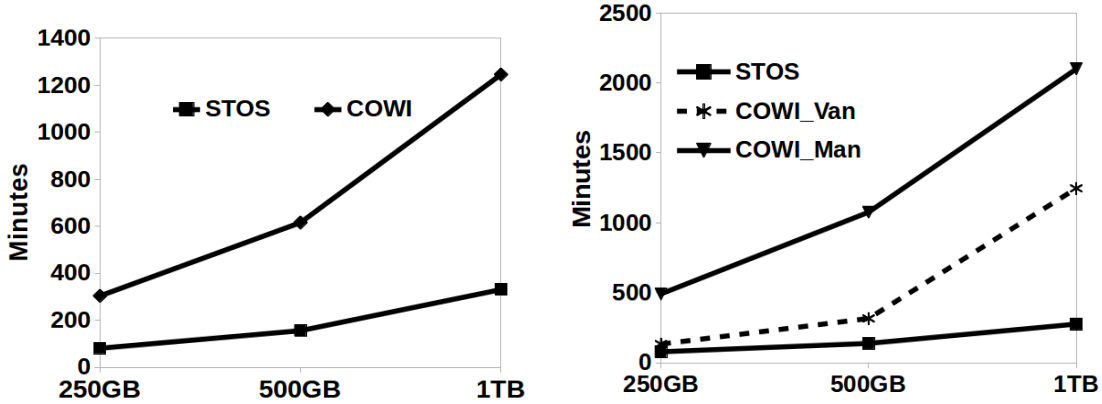
Metrics. For each method, the index building time in minutes (min) and the coefficient of variation (cov) defined as the ratio sd/E (for quantifying the load-balancing of cells) were measured, where sd is the standard deviation of the number of points stored per cell and E is the average (mean) number of points stored per cell. Moreover, the memory requirement to store the index (in megabytes) and the time to recover from failure in milliseconds (ms) were measured. Furthermore, after executing all the queries sequentially, the average query response times in milliseconds per value of k were computed. To measure the network overhead during query processing, the average number of rows (cells) retrieved and the average number of data points accessed per query were quantified.

As mentioned earlier, it is generally accepted that building indexes for competing kNN processing methods is an expensive process and is fraught with difficulties. For instance, indexing the datasets using the publicly available codes of SHadoop and Simba was not possible. Thus, the new approach was compared with COWI. Note that in the previous chapter, it was reported that COWI achieved approximately three orders of magnitude better performance than SHadoop and Simba, and COWI was reported to exhibit a better query performance than CONI (as CONI stores the index in the HBase table, thus requiring extra HBase accesses to read the index data). It is hence sufficient to compare STOS with COWI.

4.7.1 Performance Assessment: STOS vs. Index Overheads

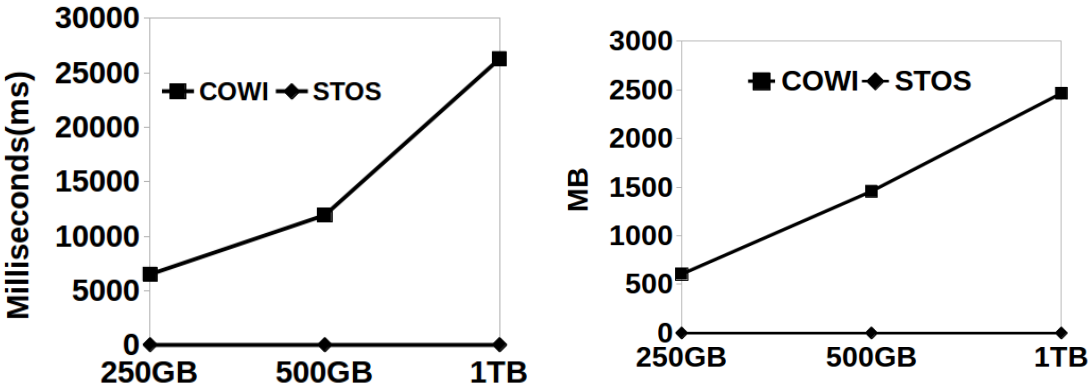
Figure 4.6(a) shows the STOS building times vis-a-vis the COWI index building times for different sizes of the AReM datasets. The COWI indexing time was roughly five times higher than that of STOS. In the literature, it is well-known that QT has a high index building time, while the uniform grid has a low index construction time [73]. Hence, the results of this experiment were as expected, confirming that this still held in STOS and COWI.

The standard HBase bulk-loading [13] technique was used to load the STOS-organised data and the COWI-indexed data into an HBase table. If the distribution of the row-keys of the HBase table is uniform, then the [13]-based loads are very efficient. Otherwise, a human expert is required to manually split the regions of the HBase table to expedite the bulk-loading process. Similarly, as shown in Figure $4.6(b)$, STOS's uniform distribution blends excellently with the existing techniques and has a better bulk-loading process than COWI: $COWI\_Man$ depicts the case when COWI's regions of the HBase table are partitioned manually, whereas $COWI\_van$ stands for a lack of manual partitioning of the regions. Note



(a) Creation/Building Time (b) HBase Bulk-Loading Time

Figure 4.6: STOS vs COWI Creation/Index Building/Storing – AReM datasets



(a) Loading Time (b) Size In Memory(Bytes)

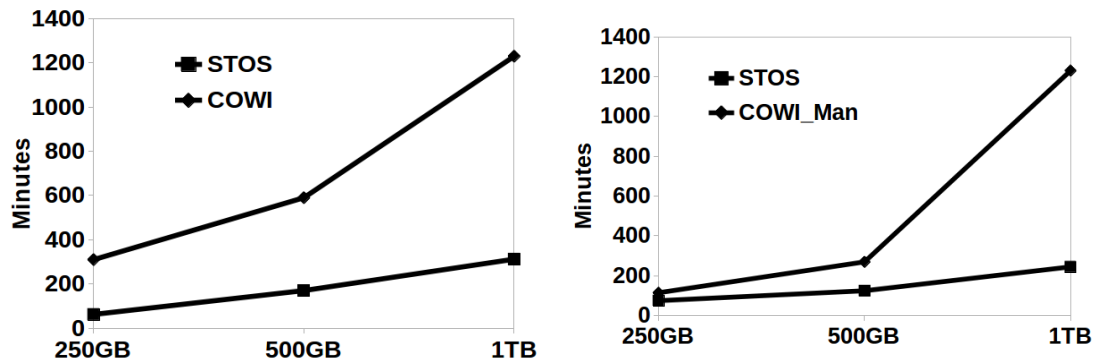
Figure 4.7: STOS vs COWI Loading – AReM datasets

that it is not always straightforward to partition a non-uniformly distributed row-keys set into equal-sized partitions manually.

The recovery time from failure (index or STOS loading time) for different dataset sizes was measured and is shown in Figure 4.7(a). In COWI, 6 s to 26 s was needed to load the index into the memory, and the index loading time increased linearly with the size of the dataset; the STOS loading time, in contrast, was constant and dramatically lower, standing at 0.014 s.

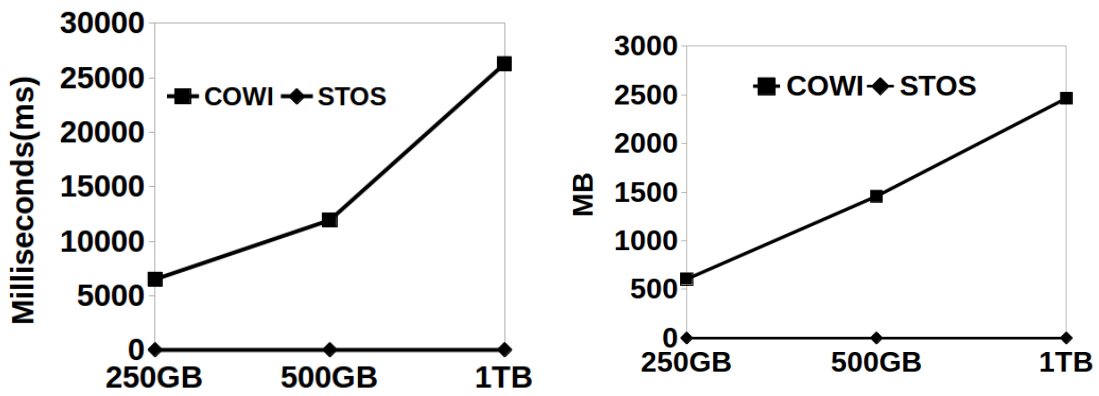
To evaluate the storage space requirements, the memory footprint of each method was measured, as shown in Figure 4.7(b). In COWI, 0.60 GB to 2.4 GB was required to store the index of the datasets. In contrast, the STOS's space requirement was (i) constant and (ii) dramatically (approximately three orders of magnitude) smaller, standing at 0.0012 GB for the different sizes of the dataset.

The index loading time and space requirements of COWI might seem to be small, in absolute numbers. However, note that the space requirement of COWI was around 0.25% of the size



(a) Creation/Index Building Time (b) HBase Bulk-Loading Time

Figure 4.8: STOS vs COWI Creation/Index Building/Storing – Istanbul datasets



(a) Creation Time (b) Size In Memory(Bytes)

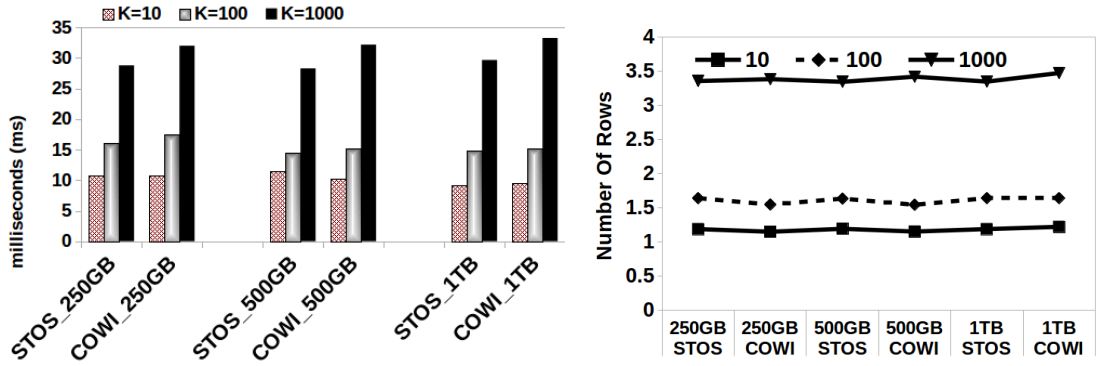
Figure 4.9: STOS vs COWI Loading – Istanbul datasets

of the dataset, and a linear increase in the index loading time w.r.t. the dataset size was observed. In other words, if the dataset increased to petabytes, then tens of gigabytes would be required. Furthermore, w.r.t. the index loading times, given the linear increase observed for COWI, for a 1-PB dataset, approximately 7 h would be needed to load the index.

The results for the Istanbul dataset were very similar, leading to largely the same conclusions to those presented for the AReM datasets. Although a detailed discussion is omitted for avoiding repetition, the STOS and COWI-index building times, HBase bulk-loading times, index loading times, and memory footprints are shown in Figures 4.8(a), 4.8(b), 4.9(a), and 4.9(b), respectively.

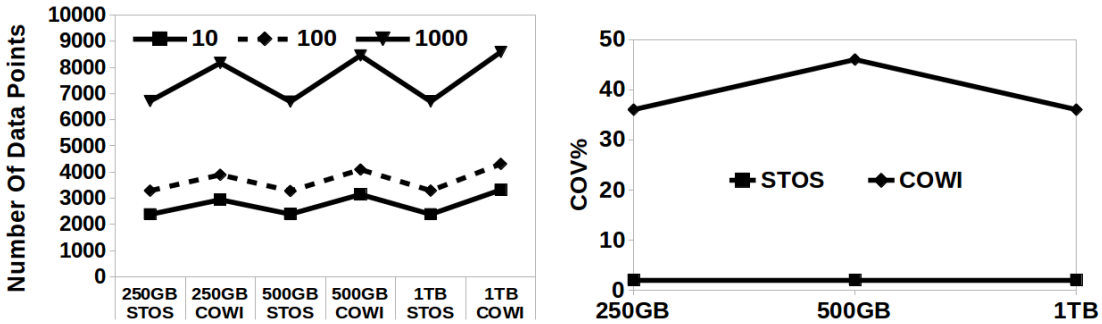
4.7.2 Performance Assessment: Query Performance

In this subsection, the query processing performance is assessed in detail. As shown in Figure 4.10(a), the kNN query response times of STOS against COWI for the 250-GB, 500-GB, and 1-TB AReM-derived datasets were compared. The kNN query response time in milliseconds



(a) Query Response Times (b) Average # of Rows Accessed Per Query

Figure 4.10: Query Processing Times and Accessed Data (Rows) – AReM datasets



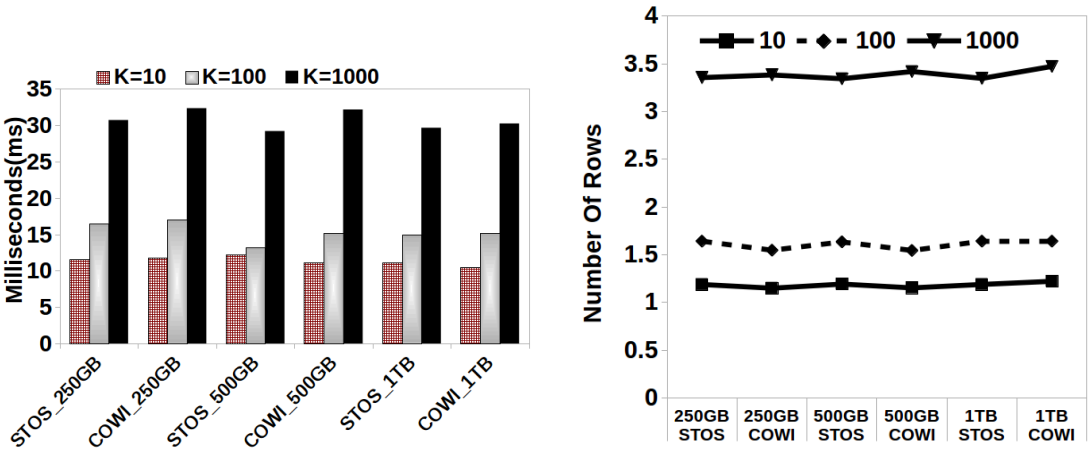
(a) Average # of Data Points Accessed per Query (b) Coefficient of Variation of Number of Data Points Per Cell

Figure 4.11: Query Processing Costs – AReM datasets

(ms) for the different values of k was measured. In STOS, the query response time varied from 10 ms to 29 ms, while in COWI, the query response time was roughly similar and ranged from 8 ms to 39 ms. The results clearly indicated that STOS had better performance than or similar performance to COWI, and both the approaches showed excellent scalability, i.e. very small query response times despite the significant increases in the dataset size.

To further assess the query processing performance, the average number of rows accessed per query. as shown in Figure 4.10(b), for different dataset sizes and the values of k were measured. STOS accessed on average 1.18 rows per query when k=10 (with, on average, 2,000 data points are stored per row) and for k=100 and k=1000, on average, 1.64 and 3.35 rows were accessed, respectively. COWI accessed on average 1.15, 1.54, and 3.34 rows for k=10, k=100, and k=1000, respectively, when on average, 2,000 data points were stored per row. This demonstrated that COWI accessed roughly the same number of rows, but in both the methods, the size of the dataset had no significant impact on the average number of rows accessed per query. The average number of rows per query was affected by the value of k; i.e. large values of k produced large query ranges.

In contrast, as shown in Figure 4.11(a), on average, STOS accessed a smaller number of data



(a) Query Response Times (b) Average # of Rows Accessed Per Query

Figure 4.12: Query Processing Costs: Query Processing Times and Accessed Data (Rows) – Istanbul datasets

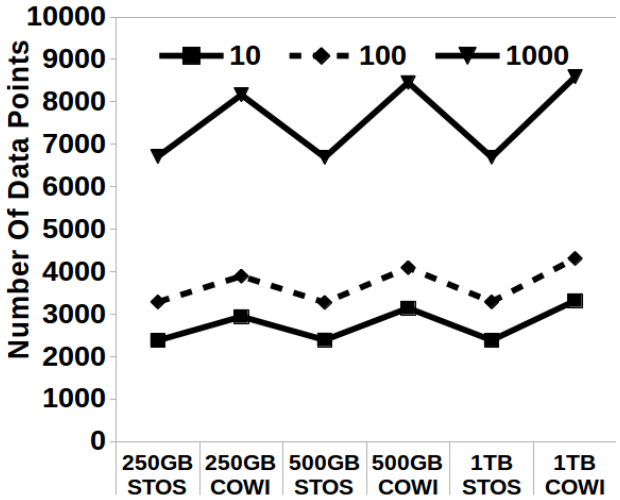


Figure 4.13: Average # of Data Points Accessed per Query – Istanbul Datasets

points per query than COWI. Initially, this seemed counter-intuitive because one can ask *how can COWI access more data points while accessing a similar number of rows as STOS?* Note that even though both the methods contained on average the same number of data points, as shown in Figure 4.11(b), the coefficient of variation (COV), which measured the variation around the mean number of items per row across the rows, in STOS was 2%, while in COWI, it was between 36% and 45%. This presented independent evidence as to the ability of STOS to partition the space equitably, with cells having nearly equal sizes. This was significant, as the partition sizes affected directly (i) the load balancing among the processes tasked with building the tree indexes or STOS itself and (ii) the query processing times. Therefore, the STOS building processes were more robust (less likely to hang during MR), and the query processing times were more predictable.

The query performance results for the Istanbul dataset were very similar; Figures 4.12(a),

4.12(b), and 4.13 show the query response times, average number of rows accessed per query, and the average number of data points accessed per query, respectively. In conclusion, both the approaches in general scaled very well w.r.t. the average number of rows accessed per query with increasing dataset sizes, with STOS having a clear edge.

4.7.3 Performance Assessment: STOS in High Dimensions

In a related work, the largely held view that QT indexes are not appropriate for datasets with high data dimensionality is discussed; this is because of the fact that QT divides a cell into 2^d sub-cells and creates a considerably large number of nearly empty cells in a high-dimensional space. Hence, during the query processing, most of the points in the QT will be accessed, and the performance is similar to that of an exhaustive search.

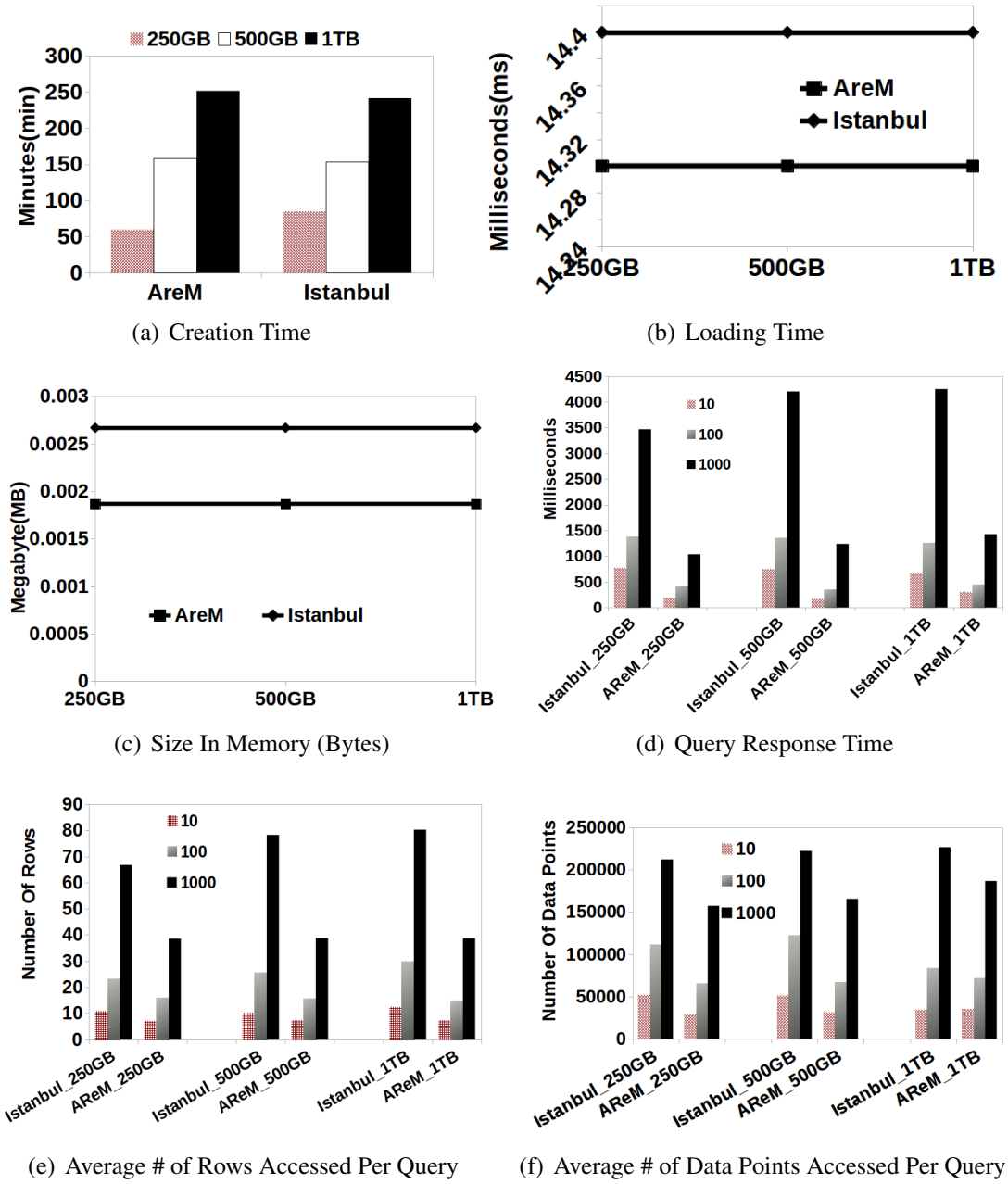
In this work, the behaviour of STOS when the data dimensionality increased was assessed. As shown in Figure 4.14(a), the STOS building time is affected by the six dimensions of the AReM dataset and the nine dimensions of the Istanbul dataset; note that the creation time of STOS was not affected by the number of dimensions, as the times required for the 6-d, 9-d, and 2-d datasets were quite similar. In contrast, the index loading time (Figure 4.14(b)) and the memory footprint (Figure 4.14(c)) were not affected by the size of the datasets. A slight increase in the memory requirement, from 1.8 KB to 2.6 KB, as the dimension increased from 6-d to 9-d was observed; also, a slight increase in the index loading time as the number of dimensions increased was observed.

As shown in Figure 4.14(d), the query response time of the AReM dataset ranged from 166 ms to 1425 ms, whereas for the Istanbul dataset, it ranged from 773 ms to 4246 ms. Again, the query response time did not show a significant change with a change in the size of the datasets, but the number of dimensions and the different values of k significantly influence the query response time, as expected. Similarly, Figures 4.14(e) and 4.14(f) illustrate the average number of rows and data points accessed per query, respectively. Note that as the number of dimensions increased, the average number of rows and data points accessed per query increased significantly; however, STOS managed to process kNN queries on average from hundreds of milliseconds to a few seconds for the 6-d and 9-d datasets.

4.8 Conclusions

For processing kNN queries efficiently, thus far, the prevalent practice by almost all of the state-of-the-art approaches is building a tree-based global and/or local index over a large-scale dataset that is stored in a distributed file system. In general, tree-based index methods have good performance but have a high storage cost and high index building time and could

be challenging to implement, particularly in parallel/distributed systems. By avoiding these problems, the principal contribution of this work, STOS, is a new method of organising and structuring a large-scale low-dimensional dataset for efficient kNN query processing. STOS transforms the distribution of a dataset into a joint uniform distribution and partitions the dataset, based on the uniform distribution, using the simple uniform-grid indexing method. When a dataset has a joint uniform distribution, the simple uniform-grid indexing method has several advantages. For example, it has (i) a small and constant memory requirement that does not increase with a dataset; (ii) relatively lower indexing time; (iii) an excellent performance; (iv) high applicability in high dimensions (in up to 10-15 dimensions); (v) excellent space utilisation (almost an even distribution of data elements across the cells); and (vi) ease of implementation. Accordingly, STOS can do away with memory-hungry indexes requiring a state with a tiny memory footprint. The new method enjoys memory requirements that represent an improvement over those of the state-of-the-art methods by up to six orders of magnitude. Additionally, the STOS memory footprint remains (nearly) constant as the dataset sizes increase, unlike traditional tree-based indexes. Furthermore, the times required to build STOS are smaller by several orders of magnitude than the traditional tree-index building times. At the same time, STOS can access and transfer very small data chunks during query processing, thus achieving very small query processing times. The above facts are critical in ensuring even higher overall kNN query processing efficiency and scalability. The viability of STOS and substantiated the above claims by using extensive experimentation over several real-world (big) datasets of various dimensionalities.



(a) Creation Time

(b) Loading Time

(c) Size In Memory (Bytes)

(d) Query Response Time

(e) Average # of Rows Accessed Per Query

(f) Average # of Data Points Accessed Per Query

Figure 4.14: STOS vs High-dimensional Data – Istanbul (9-d) and AReM (6-d)

Chapter 5

Estimated kNN and Probabilistic Self-Organising kNN Regression

5.1 Introduction

In the previous chapters, we explored the efficient computation of exact kNN in a big data environment. The main pillar that ensures the scalability of kNN is the access to a small but relevant subset of a large-scale dataset. To this end, COWI, CONI and STOS were proposed as these methods have achieved a performance gain of several orders of magnitude as compared to the state-of-the-art methods. In the previous chapter, the superiority of STOS to COWI and CONI was proven by running extensive experiments. However, with an increase in the dimensionality of a dataset, the query response time of all the state-of-the-art methods -including STOS- increases, a well-known problem. The main aim of this chapter is, therefore, to propose a solution whose query response time does not depend on the dimensions of a dataset.

Unable to solve the exponential dependency of query response time on the dimension, no efficient solution exists [60] and have proposed approximated kNN for high dimensional space. AkNN retrieves k-NN (not possibly the nearest data elements to a given query) from large-scale datasets. The distance of the retrieved data elements is at most m times the distance from the query to its nearest points. The rationale behind AkNN is that an approximate nearest neighbour is almost as good as the exact one, particularly when the distance measured accurately captures the notion of user quality; then, small differences in the distance should not matter [9]. Recently, statistical learning-based AkNN has attracted considerable attention, and [82] provides a survey of the most popular methods.

Drifting away from the AkNN, a new perspective of computing kNN is proposed. The proposed method -called estimated kNN- adapts to the design philosophy of AKNN. However,

estimated nearest neighbour is almost as good as the exact one, especially when the distance measured accurately captures the notion of user quality, then small differences in the distance should not matter. Accordingly, estimated kNN provides the average difference in distance between the estimated nearest neighbours and the corresponding actual nearest neighbours with some degree of confidence (without retrieving anything from a dataset); hence, a user can decide whether the average distance accurately captures the notion of his/her quality.

As the proposed method estimates the nearest neighbours by using the space transformation technique of STOS, introduced in the previous chapter where a case when a user is not satisfied with the average distance of the estimated nearest neighbours from the actual neighbours, STOS can be invoked to compute exact kNN particularly when the number of dimensions is $\leq 10 - 15$. As will be shown later in the experimental results, estimated kNN not only predicts the nearest neighbours that are located a short distance away from the actual nearest neighbours but also has several orders of magnitude lower query response time than STOS: this is more significant particularly in case of higher-dimensional data. In addition to that, as the proposed method computes estimated kNN using the space transformation technique, explained in the previous chapter, it has a minute memory footprint and as such can be deployed to a client-side.

Motivating Example

Consider a fictitious tourism application that stores a survey of almost all the restaurants in the world and recommends k nearest restaurants to a user based on specific criteria (dimensions): such as spatial location, average food quality, average service quality, average food price, average waiting time, average value for money, and average spending per customer. Assume STOT is used to index the restaurant data. When a user executes kNN to a given query, instead of sending the query to a server and waiting for a response, an estimated kNN can be computed locally at the client-side. The locally computed answer would look like this: the first closest restaurant is located at this particular coordinate or within x radius from the estimated location, has average food quality of z $\pm$ y and so on. When the prediction interval is small, and the level of confidence for actual result to be located within the interval is high, a user might be satisfied with the estimated results; otherwise the user can get exact results from the server. When a user is satisfied with the estimated results, such a design is intended to achieve two distinct advantages: (i) query response time of the estimated results should not have exponential dependency on dimensions; and (ii) should play a significant role in reducing coordinator (server) workload and ease network traffic; as such contributes on improving query response time.

In contrast, in this part of the thesis, based on the space transformation technique, another novel kNN-based regression is proposed. The proposed method is affectionately called prob-

abilistic kNN regression. Similar to the estimated kNN, probabilistic kNN regression can be used either as a stand-alone solution or in parallel with STOS.

A traditional kNN regression retrieves kNN from the underlying dataset and computes the average value of a target RV of the kNN data elements. The distance between a query and the elements of a dataset is computed using the predictor (observed) RVs.

Unlike the kNN regression method, the proposed probabilistic kNN regression, without accessing a dataset, predicts k plausible values of a target RV by using the space transformation technique and then, computes the average of the k predicted values to provide the final answer.

As probabilistic kNN does not access a dataset and has no disk I/Os, it has a small query response time; moreover, experimental results showed that probabilistic kNN had a small prediction error. Lets recall that all the statistical parameters that are needed for space transformation are designed to be stored in a small memory footprint. Consequently, by deploying such parameters on the client-side, the estimated kNN and probabilistic kNN regression queries can be executed at the client side locally. Such a design not only has a significant advantage in reducing the workload of a coordinator (server) but can also ease the network traffic and thereby improve the query response time.

After conducting a thorough literature review, the estimated kNN and probabilistic kNN regression are the first approaches toward *dataless* kNN processing in the big data environment. Hence, this study recommends that future research examine dataless big data processing for other complex queries such as join and range queries in the big data environment.

5.2 Contribution

The salient contributions of this chapter are as follows:

- A novel approach for estimating kNN and predicting kNN regression on the basis of the Space Transformation approach. Both the methods facilitate query processing as though the underlying datasets were uniformly distributed.
- Extremely low query response time, several orders of magnitude smaller than that of the exact kNN query processing methods and the kNN regression methods.
- The query processing time is practically independent of the number of dimensions of a dataset.
- High prediction accuracy.

- Small memory footprint and the memory space requirement does not exponentially increase with the number of dimensions.
- Can easily run on the client-side, thus reducing network I/Os and the workload of a coordinator.
- *Dataless* query processing, with no disk and network I/Os, irrespective of the size of a large-scale dataset.

The rest of this chapter is organised as follows: section 5.3 reviews the related work, section 5.4 presents the methodology, 5.5 reports the experimental evaluation and 5.6 concludes the chapter.

This chapter will be submitted to 2020 IEEE International Conference on Big Data (IEEE BigData 2020):

- A. S. Cahsai, C. Anagnostopoulos (2018) *Scaling-out kNN Regression via Probabilistic and Estimated kNN Query Processing*, 2020-21 IEEE International Conference on Big Data (IEEE BigData 2020-21])[will be submitted Oct 2020-21]

5.3 Related Work

To the best of the author's knowledge, [36] is the only related method that uses a copula -space transformation- for kNN-based imputation; the data in [36] combine the field data from forest inventories and the auxiliary information for forest resource estimation at various geographical scales. The [36] constructs a model, consisting of a canonical vine copula, from the empirical data, and hence, new samples are generated from the model and used for kNN predictions. Note that [36] does not transform the distribution of the data into a joint uniform distribution.

The approach, [36], is different from probabilistic kNN, proposed in this thesis, in several ways. First, probabilistic kNN transforms an arbitrary distribution of a dataset into a joint uniform distribution using an independence copula but [36] builds C-vine copula to capture the dependence between the RVs of a dataset and generates sample data on the basis of a model consisting of the C-vine copula. Second, probabilistic kNN regression generates only k values of a target RV, irrespective of the size of a dataset and this makes probabilistic kNN suitable for predicting a value of a target RV in a large-scale dataset. In contrast, [36] generates copula samples of the same size as that of the original reference data; thereafter, the copula samples are used for the nearest neighbour imputation. Hence, [36] is not suitable in the big data environment.

5.4 Methodology

5.4.1 Estimated kNN

When tree-based multi-dimensions indexes are used for indexing a high dimensional dataset, a previous study [47] shows that a linear increase in dimension has an exponential increase in query response time and memory space requirements. In the previous chapter we have shown that even though STOS manages to remove the exponential dependency of memory space requirements on dimension, query response time still increases with dimension; see the experimental result in section 4.7. In STOS, the average number of rows accessed per query increases with dimension.

When processing a kNN query in a large-scale dataset, to solve the problem of accessing more rows as the dimension of a dataset increases, a novel solution called estimated kNN query processing is proposed in this chapter. Without accessing the a large-scale dataset, the proposed approach provides an estimated answer to a given query. The proposed method also asserts that an actual kNN is located within a given distance r from an estimated kNN with a φ level of confidence.

The estimated kNN query processing method can be used either as a stand-alone solution or in conjunction with STOS. When it is used with STOS, a user can be given an option to either wait a relatively long time to get an exact answer or promptly get an estimated kNN results. Along with a kNN query, a user can also submit distance difference between the actual and the estimated nearest neighbour that the user is willing to tolerate; then the proposed method provides estimated results and a φ level of confidence, which states that $x\%$ of the actual results are located within a user-defined distance from the estimated results.

Similar to STOS, the estimated kNN processing has the following building steps: (i) clustering a large-scale dataset using GMM, (ii) computing the mean vector and the covariance matrix of each cluster, (iii) computing the cdf, cumulative distribution function of each RV of each cluster, and (iv) computing the inverse cdf of each RV of each cluster.

Rationale behind Estimated kNN Query Processing

Before explaining why the estimated kNN processing works, it is crucial to understand why exact kNN processing methods access more rows in case of high-dimensional data. Exact kNN processing approaches such as STOS access more rows because of the following two reasons:

(R1.) Recall that computing exact kNN consists of two steps: (i) identify the closest cell (partition) and compute an initial kNN; and (ii) after an initial kNN answer is computed,

a circle centred at the query with a radius defined as the distance between the query and the kth data element of the initial kNN answer is drawn. Then any neighbouring cells that overlap with the circle are accessed to compute the final kNN answer. Consequently, if the circle overlaps with only one cell, then only the data elements that reside in this cell are retrieved to answer the kNN query. However, when the circle overlaps with, for instance, s cells per dimension, in all s^d cells are accessed, where d denotes the number of dimensions of the dataset. Thus the average number of rows accessed per query depends on dimensions; for practical example, please refer to the experimental results of fig. 4.14(e) in the previous chapter.

(R2) This is related to the curse of dimensionality. The distance between data elements increases with the dimension. For a simple clarification, without any loss of generality, assume that a dataset that has a standard uniform distribution has n number of data points. When computing kNN, a fraction of $(k/n)^{(1/d)}$ of the total volume of the domain space is required to be accessed to retrieve the k data elements. When d is large and n is small, the fraction of the total volume that might be required to be accessed increases and might reach to the point where the entire dataset is exhaustively searched. This problem can be alleviated when n is large whereas k and d are relatively small; see fig. 4.14(e). This part of the thesis, the assumption is that there is there is a very-large-scale dataset in which this particular problem does not occur.

Accessing only one row per a query can be an ideal solution for the first problem. However, once a dataset is partitioned, and the boundaries of the cells are determined, it is impossible to access only one cell without losing some data elements that can be part of the exact kNN answer. To this end, an estimated kNN answers are proposed to avoid the dependency of query response time on dimension.

The proposed method acts as though the dataset were indexed online during the query processing time. Thus, during query processing, an imaginary cell centred at **q** (the query) with a sufficiently large width to include all the potential kNN is created. Theoretically, such a design allows accessing only one cell per query, but practically creating an index online is in-feasible because of the high index building time.

Recall that the space transformation technique introduced in the previous chapter transforms a random distribution of data into a standard uniform distribution. In the uniform space, all cells contain an approximately equal number of data elements and have the same size. Furthermore, data elements that belong to the cell of the uniform space are randomly distributed in the cell.

In the uniform space, once the size of a cell that contains k data is defined, k data elements can be randomly generated to fill up the cell. Thus, the distribution of the randomly generated data elements and the actual data elements have uniform distribution in the cell. Afterwards,

intuitively, when the randomly generated data elements are transformed into the original data space, they are expected to have similar values to that of the original data elements. Hence, the question is, how can we determine the size of a cell for a given query.

Note that all the dimensions of a cell, in the uniform space, have equal width. Therefore, a width of a cell, centred at a given query, can be defined using the following equation:

$$w = (k/n)^{(1/d)}$$

where n is the total number of data elements and d is a number of dimensions of the dataset.

Once the size and location of the cell are determined, data elements that lie within the boundaries of the cell can be generated easily at random. However, these data elements have to be transformed to the original space in order to estimated the kNN. As the distance between the uniform space and original space is not invariant, the query in the original space might not be located at the centre of the cell that is transformed into the original space.

Accordingly, to improve the estimated kNN answer, an initial estimated kNN query is computed from the data elements transformed to the original space. Accordingly, a circle centred at the query with a radius, ρ, (defined based on the distance between the query and the kth data element of the initial estimated kNN answer) is drawn in the original data space. Intuitively, using the following four steps the accuracy of the estimated kNN can be improved: (step-1) transforming the circle or a minimum bounding hyperrectangle (MBR) that encompasses the circle to the uniform space; (step-2) in the uniform space, randomly generating data elements to fill up the transformed MBR; (step-3) transom the randomly generated data elements into the original domain space; and (step-4) compute the finale estimated kNN from the data elements transformed to the original space in step-3. The number of data elements that lie within the transformed circle(or MBR) of the uniform space, denoted as (k^{est}), can be computed using the following equation:

$$k^{est} = Area(mbr^{uni}) \times n$$

where $Area(mbr^{uni})$ is the area of the MBR (or the circle) transformed into the uniform space and n is the total data elements in the dataset. Thus, k^{est} data elements must be transformed into the original space in order to compute a final estimated kNN.

The discrepancies between the estimated and the actual kNN are computed by the root mean square error (RMSE). The proposed method provides an estimation summary which states that $x\%$ of the estimated kNN will be within a radius,r, from the corresponding actual kNN results. The radius is computed based on RMSE * h, where h is related to $x\%$ of the estimation summary and is obtained from the 1-$(x/100)$ quantile of the Chai-square, $\mathcal{X}^2$, distribution with a df degree of freedom; assuming that the measurement errors have a Multivariate

Gaussian distribution. When the RMSE is small and the level of confidence ,$x\%$, is high, the estimated kNN might be as good as the actual kNN.

5.4.2 Probabilistic kNN Regression

Based on the distance (or similarity) metric, a kNN regression predicts the unknown target variable of a query by retrieving kNN from an underline dataset and computing a weighted average of the kNNs' target RVs. Similar to a kNN, a query response time of kNN regression has an exponential dependency on dimensions. Again to remove such dependency, a novel approach, probabilistic kNN regression, is proposed in this chapter. The proposed method estimates values of target RVs of k nearest data elements without retrieving any data element from the back-end. Probabilistic kNN regression exploits the data space transformation technique, explained in the previous chapter, to predict the unknown target RV of a query; the rationale behind the proposed approach will be described next.

Rationale behind Probabilistic kNN Regression

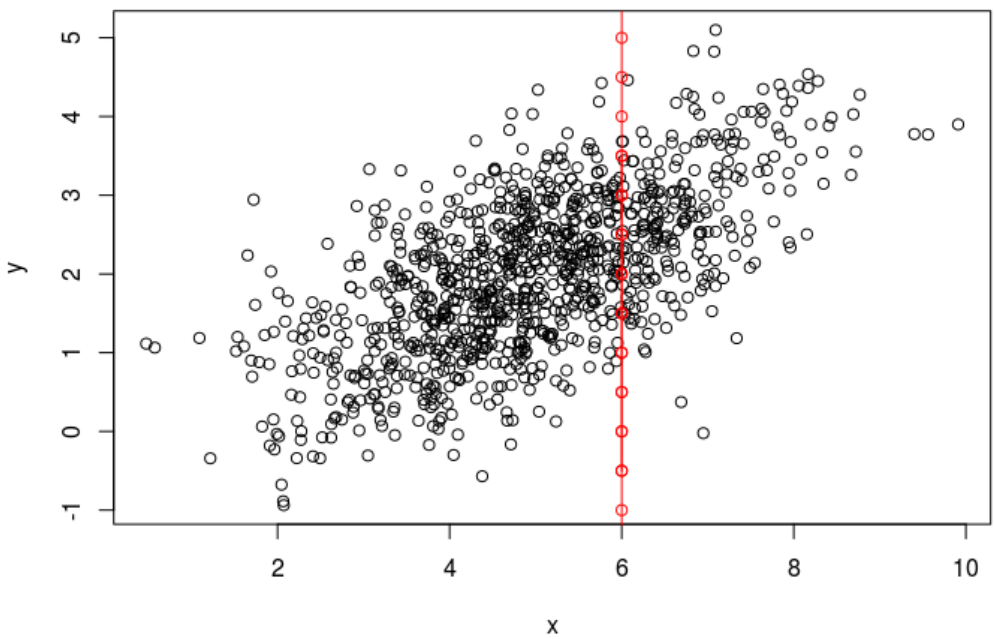


Figure 5.1: Original Space

Again, before explaining the rationale of probabilistic kNN regression, let us quickly revisit how the traditional kNN regression works. The kNN regression first finds kNN by using

the observed (predictors) RVs of the query and computes the average of the target RV of the retrieved k data elements to predict a final answer. For example, without any loss of generality, consider that a two-dimensional domain space and a target variable t of a vector $\mathbf{q} = (6, t)$; so assume that one is interested to find the value of t by running the traditional kNN regression algorithm over a fictitious dataset, shown in fig. 5.1. Intuitively, the value of t can be any value of the y-coordinate of the red line depicted in fig. 5.1; thus, reasonably, the traditional kNN regression attempts to compute t by averaging k values of the y-coordinate, i.e, black dots (data points) that overlap the red line in the given figure.

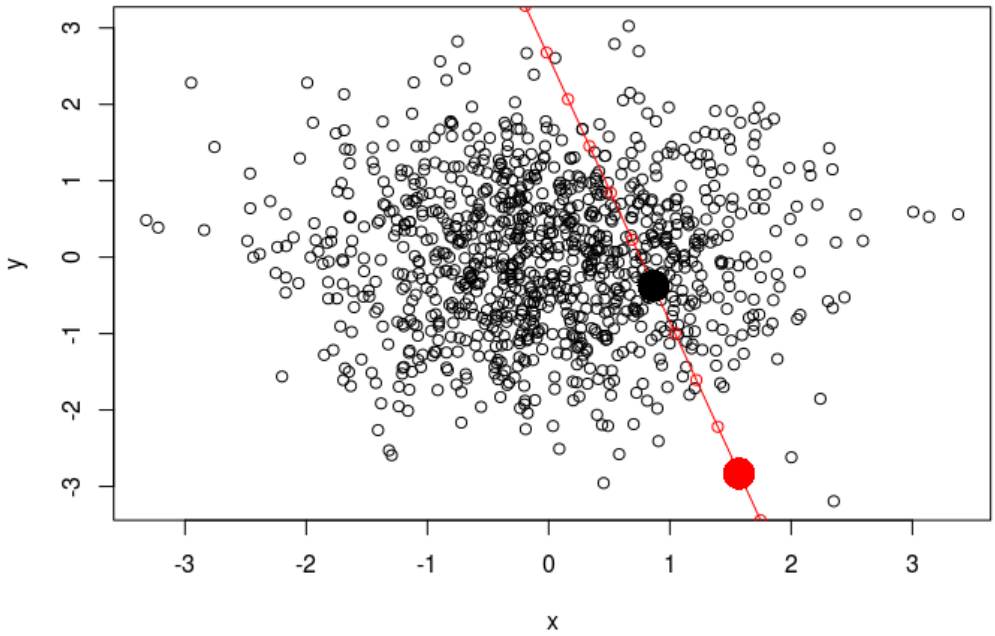


Figure 5.2: Independent Space Time

To this end, if one can estimate the values of the target RV (y-coordinate) of k data elements that overlap the red line of fig. 5.1 without retrieving data elements from a dataset, then the estimated values can be averaged to answer a kNN regression query. Therefore, the main question of interest is how to determine the plausible values of the target RV to a given query with high accuracy.

This question of interest can be answered using the data space transformation technique. However, before describing the answer mathematically, it is much easier to explain the solution using a simple example. Let us consider the red line on which the relevant data points to $\mathbf{q} = (6, t)$ lie in the original domain space; please refer to fig. 5.1. When the dataset is transformed to the independent and uniform spaces, the location of the red line in the independent and uniform spaces is shown in fig. 5.2 and fig. 5.3, respectively.

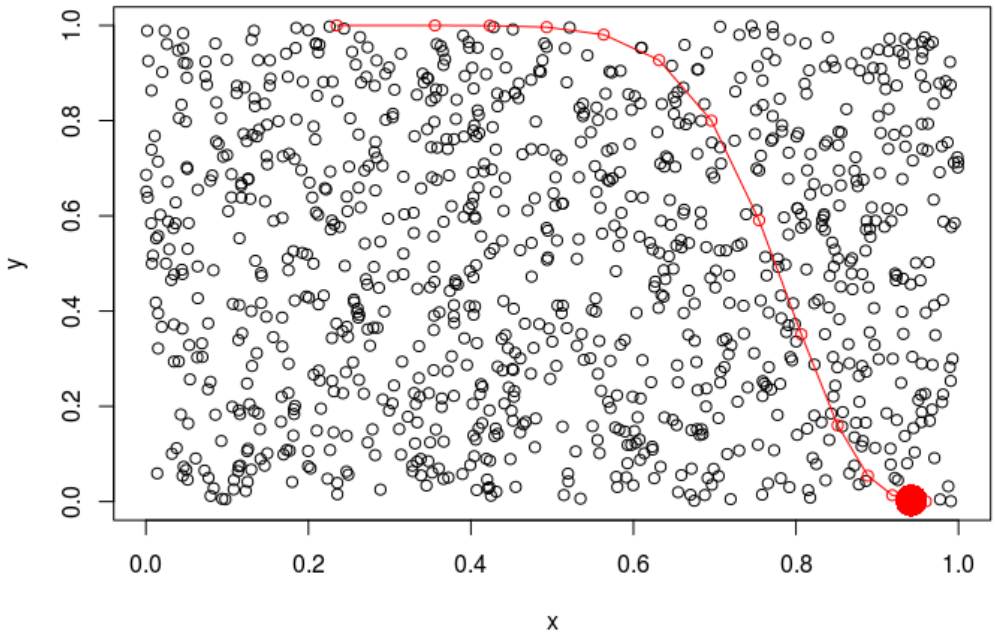


Figure 5.3: Uniform Space

Note that, in the uniform space, not only the entire transferred data but also values of the y-coordinate of the semi z-shaped red line (produced by transforming the red line of the original space) has (i) a marginal standard uniform distribution as shown in fig. 5.3; (ii) values of the y-coordinate of the diagonal part the semi z-shaped red line *spread* evenly between 0 and 1; and (iii)when this diagonal line transformed back to the original space, it corresponds to part of the vertical red line of the original space, which overlaps with the dense region in which the actual data points (black dots) are located (see fig. 5.1). Similarly, when the values of the RV y that are *concentrated at both ends* of the semi z-shaped red line of the uniform space are transformed back to the original space, the corresponding data points are located on the upper and lower part of the vertical red line of the original space that do not overlap with the black dots (actual data elements).

For example, in fig. 5.1, consider a point (6,0) that lies on the part of the red line that does not overlap with the region in which the actual data points are located; intuitively, the value of the RV y that corresponds to the point, (6,0) in the original space, is close to zero in the uniform space; please refer to the big red dot in fig. 5.3.

To this end, values of a target RV that lie in part of the red line that overlaps with the region in which the actual data points are located is considered as *relevant* to the given query and are evenly spread between 0 and 1 in the uniform space. However, the values of the target RV that belong to part of the red line that does not overlap with the region of the actual data

points are considered *irrelevant*. Irrelevant values are concentrated at the opposite ends of the semi z-shaped red line of the uniform space. This is because values that differ significantly from the rest of the data elements, broadly speaking, are considered outliers. The cdf of outliers is usually close to either 0 or 1; recall cdf is used to transform RV of the independent space into a standard marginal uniform distribution as explained in the previous chapter.

Assuming that the values of the predictor RV, x-coordinate of the z-shape line, is known, one can estimate the values of the RV y by randomly generating numbers between 0 and 1. When the estimated values of the target RV are transformed to the original data space, most of the transformed data points lie on the relevant part of the red line. This implies that the data transforming technique can be used to estimate the output of a kNN regression to a given query. In this thesis, determining the value of the target RV using data transformation technique is referred to as a probabilistic kNN regression.

All the points that have been discussed so far are the core pillars of the proposed probabilistic kNN regression. Armed with this knowledge, methodology of the proposed method is described next.

5.4.3 Methodology of Probabilistic kNN Regression

In the previous chapter, it was explained that the data elements of the original data space could be transformed into independent space by using eq. (4.3). However, transforming a data element into the independent space requires all the values of the RVs (dimensions) of a data element to be known in advance; in the case of the kNN regression, the value of a target RV is not known in advance, and hence cannot be transformed into the independent space without filling the target RV with some values.

The first step of the proposed method is to set the value of a target RV of the query to zero; thus, the given query can be transformed into the independent space by using eq. (4.3). This implies that initially, the proposed method transforms the value of a target RV of the kNN regression query, **q**, into the independent space by using eq. (5.1).

$$t_{ind}^{init} = \sum_{i=1}^{d} \boldsymbol{A}_{di}(q_i - \boldsymbol{\mu}_i) \tag{5.1}$$

where q_i refers to the ith dimension of **q**, $q_d = 0$ (assuming that the target variable is in the dth dimension), t_{ind}^{init} is a value of the target RV of **q** in the independent space, $\boldsymbol{\mu}_i$ represents the mean value of the ith RV of **q**, $\boldsymbol{A}$ indicates the whitening matrix a cluster, and d denotes the number of dimensions of **q**;

For instance, consider the $\mathbf{q} = (6, t)$ used in the previous examples. To transform the query **q** into the independent space, the proposed method first replaces t by zero and then applies

eq. (4.3). The transformed point is shown in fig. 5.2 as a big red dot. As the value of the target RV of **q** in the original space is arbitrarily set to zero, the corresponding value t_{ind}^{init} of the big red circle in the independent space is located outside the range in which the actual data points reside.

The main focus of the proposed method is to move the arbitrary location of the target RV of the independent space into the *relevant* region by exploiting the uniform distribution of the corresponding RV of the uniform space. For example, the main idea is to move the big red dot shown in fig. 5.2 into the location of the big black dot shown in the same figure. To do so, the proposed method has the following three steps. (step-1) Generates randomly a number, t_{uni}, between 0 and 1 in the uniform space. (step-2) Transforms t_{uni} into t_{ind} of the independent space by using the inverse cdf function of the target RV. Recall that the probability of t_{ind} to be located within the relevant region is high because only values of t_{uni} close to either 0 or 1, when transformed back into the independent space, fall outside the relevant area. (step-3) the value of t, the target RV of the query in the original space, (denoted as q_d), is computed on the basis of the following equation:

$$(t_{ind} - t_{ind}^{init}) = ((\sum_{i=1}^{d-1} \boldsymbol{A}_{di}(q_i - \boldsymbol{\mu}_i) + (\boldsymbol{A}_{dd})(q_d - \boldsymbol{\mu}_d)) - ((\sum_{i=1}^{d-1} \boldsymbol{A}_{di}(q_i - \boldsymbol{\mu}_i) + (\boldsymbol{A}_{dd})(-\boldsymbol{\mu}_d))$$
$$(t_{ind} - t_{ind}^{init}) = (z + (\boldsymbol{A}_{dd})(q_d - \boldsymbol{\mu}_d))) - (z + (\boldsymbol{A}_{dd})(-\boldsymbol{\mu}_d))$$
$$q_d = (\frac{(t_{ind} - t_{ind}^{init})}{\boldsymbol{A}_{dd}})$$

where t_{ind}^{init} denotes the initial value of the target RV of **q** computed using eq. (5.1), $\boldsymbol{\mu}_d$ indicates the mean value of a target RV of **q**, $(\boldsymbol{A}_{dd})$ represents the variance of the target RV of **q**.

5.5 Experiments

5.5.1 Experimental setup

The Experiments for computing the exact kNN were run on a five-node cluster; each node is a Dell R720 server with four Intel Xeon CPUs (eight cores each), 64 GB RAM, and 2 TB disk space; the experiments for the estimated kNN and probabilistic kNN regression (for both exact and estimated) were run an ordinary Linux machine. In the experiments, performance the proposed approaches were compared to exact kNN approaches.

5.5.2 Datasets

The 6-dimensional and 9-dimensional datasets used in the previous chapter (see section 4.7) were reused in a part of the experiments. The kNN results obtained in the previous chapter were also used as the ground truth when comparing the accuracy of the estimated kNN results. Furthermore, the following three real-world datasets from the UCL machine learning data repository were used: (i) The Population Biology of Abalone Haliotis species in Tasmania (coined Abalon) [83], (ii) Combined Cycle Power Plant (CCPP) [78], and (iii) Air Quality (Air) [29].

Query Workload and Performance Metrics

From the datasets used in the previous chapter, 10K queries were randomly selected from each file. For the remaining three datasets, i.e. Abalon, CCPP, and Air, 100 queries were randomly selected from each file. The experiments were conducted to evaluate the performance of the proposed methods for three different values of k, such as $k \in \{10, 100, 100\}$.

During the evaluation of the estimated kNN queries, the query response time was measured in milliseconds; the RMSE was used to quantify the prediction error, the radius (the average distance between the estimated NN and the actual NN) was measured using the Euclidean distance, and the actual coverage probability and the nominal coverage probability were used to measure the fraction of the actual NNs located within the given radius of the corresponding estimated NNs.

For evaluating the probabilistic k regression, MAE, RMSE, and coefficient of variation (CV) were used to compare the accuracy of the traditional kNN regression against that of probabilistic kNN regression.

5.5.3 Accuracy Assessment: Estimated vs. Actual kNN

The RMSE of the estimated kNN was computed as shown in equation 5.2:

$$RMSE = \left(\frac{1}{Q * K * D} \sum_{q=1}^{Q} \sum_{k=1}^{K} \sum_{d=1}^{D} (a_{qkd} - e_{qkd})^2 \right)^{1/2} . \tag{5.2}$$

where Q denotes the total number of queries, K stands for the value of k, D indicates the total number of dimensions of the dataset, and a and e represent the corresponding values of the actual and the estimated kNN, respectively.

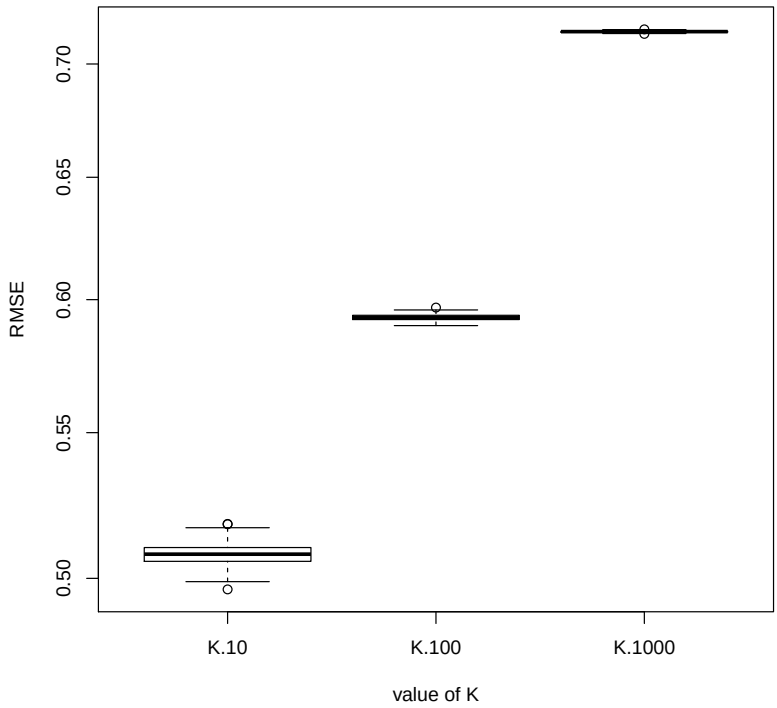


Figure 5.4: RMSE of AReM 6-dimensional dataset – 250GB.

When an estimated kNN for the given query was executed repeatedly, a small variation in RMSE was observed, as the estimated points were produced on the basis of the random generation of points in the uniform domain space. Therefore, all the experiments were repeated 100 times to quantify the difference in RMSE.

Hence, for the AReM six-dimensional 250-GB dataset, fig. 5.4 shows the RMSE of the estimated kNN over different values of k by using box-plot diagrams that capture the variation of the RMSE over the experiments repeated 100 times for different values of k. The average RMSE was 0.53, 0.58, and 0.73 for 10, 100, and 1000 values of k, respectively. In general, the variation of the RMSE was very small, and as the value of k increased, the variation decreased. Hence, these results made a significant contribution by demonstrating not only the accuracy but also the reliability of the proposed estimated kNN query processing method.

For the same dataset, the average distance between the estimated NN and the actual NN was computed as shown in equation 5.3 as follows:

$$AverageRadius = \frac{1}{Q * K} \sum_{q=1}^{Q} \sum_{k=1}^{K} d(\mathbf{a}_{qk}, \mathbf{e}_{qk}). \tag{5.3}$$

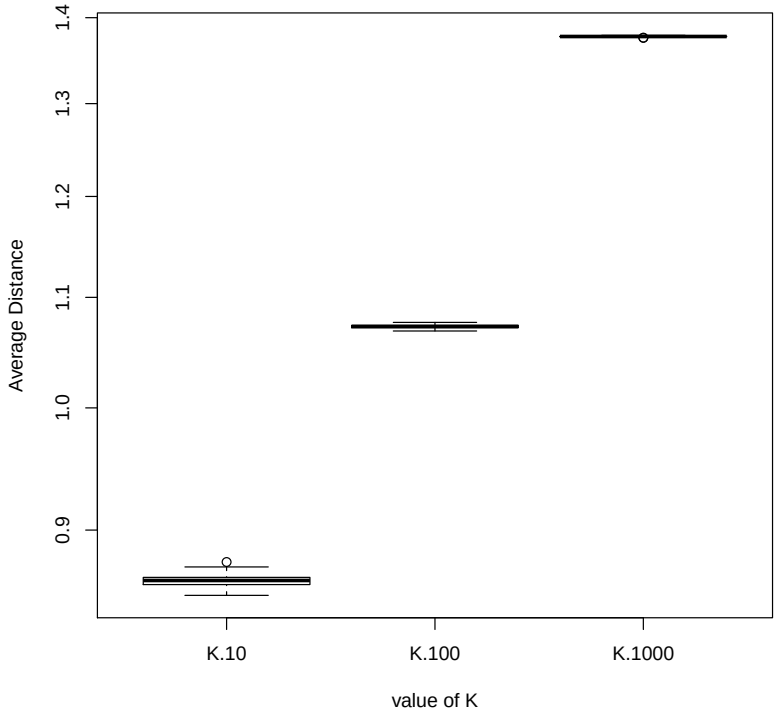


Figure 5.5: Average Distance between Actual and Estimated NN of AReM Six-Dimensional dataset – 250GB.

The average distance between the estimated NN and the actual NN for the AReM six-dimensional 250-GB dataset is reported in fig. 5.5. Again, each experiment was repeated 100 times, and no significant variation in the average distance was observed over different experiments. The average distance between the estimated NN and the actual NN varied from 0.8 to 1.37 when k varied from 10 to 1000.

As the RVs of a dataset can represent different attributes that have different measurements, the average distance (radius) between the actual and the estimated points can be difficult to interpret. To give the radius a meaningful interpretation, it can be interpreted as the average distance between the RV of the actual and the RV of the corresponding estimated data element, and not the distance between the actual and the estimated data. Thus, on average, each dimension of the actual NN was located within the average radius from the corresponding dimension of the corresponding estimated NN; for instance, as the first dimension of the dataset represented heartbeats, on average, the heartbeat of the actual NN was 0.8 minus or plus the corresponding estimated heartbeat.

Furthermore, to quantify the fraction of the actual kNN, all of whose RVs lie within the average radius from the corresponding RVs of the corresponding estimated kNN, the coverage probability (noted as Actual-CI) was adopted. Thus, the coverage probability of the AReM

six-dimensional 250-GB dataset is shown in fig. 5.6, and the results showed that the coverage probability was 90%, 91.2%, and 92% for k = 10, 100, and 1000, respectively. As the radius of k=1000 is larger than k= 10 and $k = 100$, it had a wide confidence interval and thus higher coverage probability, as expected. However, as the radius for all the values of k was not very large (see fig. 5.5), it could be safely concluded that the proposed method estimated the actual kNN with high precision.

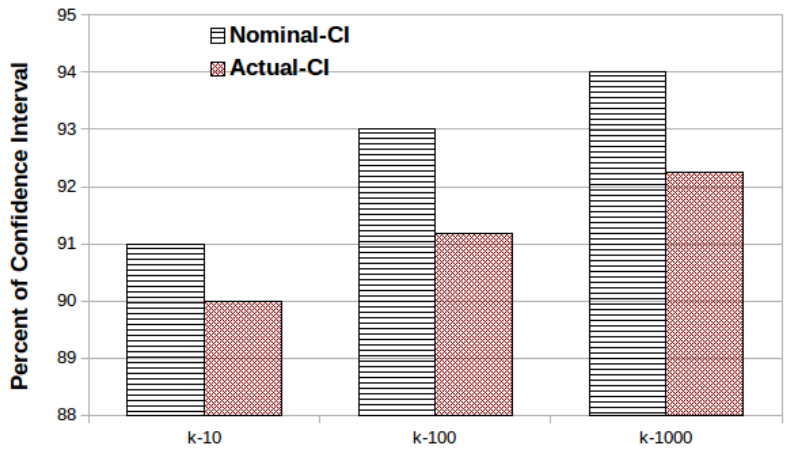
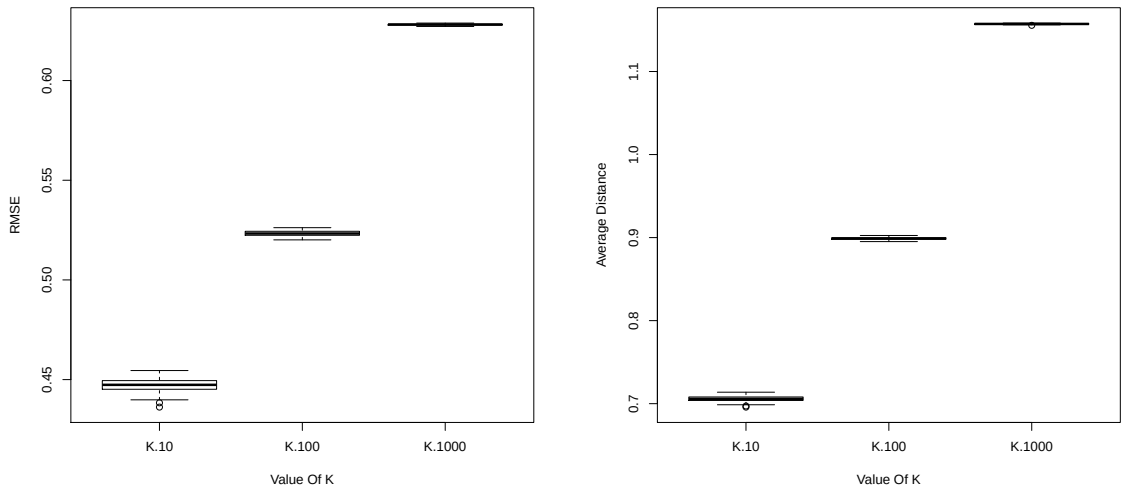


Figure 5.6: Nominal CI and coverage probability of the actual and estimated NN of AReM 6-dimensional dataset – 250GB.

Note that the radius can be defined as $radius = rmse \cdot \mathcal{X}_p^2$, where $\mathcal{X}_p^2$ is the value of the chi-square distribution at the given level of confidence.

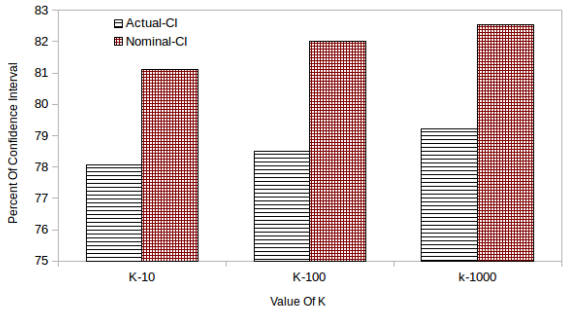
Therefore, it is not surprising that the radius and the RMSE obtained from the experiments can be used to compute $\mathcal{X}_p^2$ (loosely defined as nominal coverage probability ,denoted as nominal-CI, i.e. the fraction of the actual kNN expected to be located within the radius of the estimated kNN). Accordingly, in fig. 5.6, the nominal-CI is computed using the radius and the RMSE for each value of k and is displayed alongside the Actual-CI. As the nominal coverage probability and the actual coverage probability were relatively close, a user could provide a radius (the average difference between the actual and the estimated results that the user was willing to tolerate) and the system could notify the user the degree of confidence on the basis of the $\mathcal{X}^2$ distribution.

Similar results were observed across different AReM and Istanbul datasets that had different sizes. For discussion, the accuracy matrices of two datasets, namely the dataset generated based on AReM that had 1-TB size and the dataset created based on Istanbul that had 1-TB size, are reported. In fig. 5.7(a), fig. 5.7(b), and fig. 5.7(c), the RMSE, average distance (radius), and the nominal and actual confidence probability of the AReM dataset (size: 1 TB) are shown, respectively. The RMSE and the average distance of this dataset were smaller than those of the 250-GB dataset; this was expected because having more datasets decreased the fraction of the total volume of a cell on the uniform space within which the random



(a) RMSE

(b) Average Distance (radius)



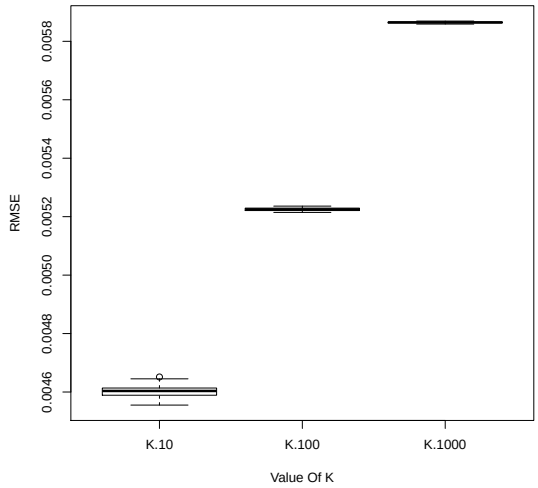
(c) Confidence Interval

Figure 5.7: Dataset AReM: Size 1 TB

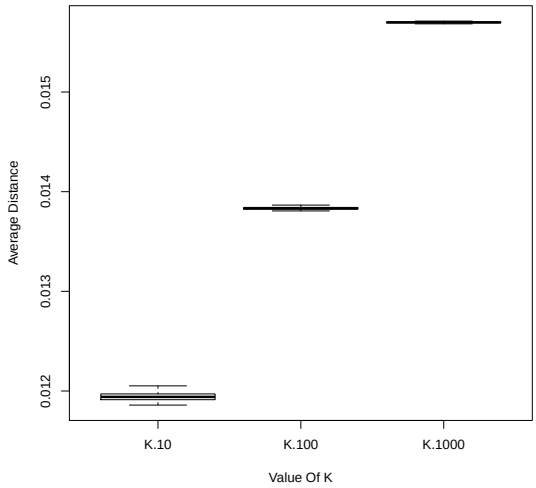
data elements were generated. In contrast, as shown in fig. 5.7(c), the difference between the actual and nominal coverage probability was small. This demonstrated that the radius computed on the basis of the $\mathcal{X}^2$ distribution could accurately predict that fraction of the actual kNN located within the user-defined distance from the estimated kNN. Furthermore, fig. 5.8(a), fig. 5.8(b), and fig. 5.8(c) show the RMSE, average distance, and the nominal and actual coverage probability of the 1-TB Istanbul dataset. The same conclusion was drawn as that for the above datasets.

5.5.4 Performance Assessment: Estimated vs. Actual kNN

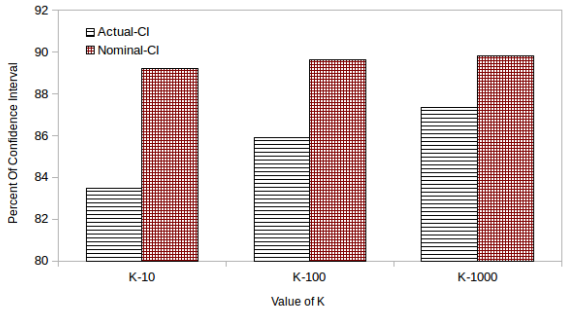
In fig. 5.9(a) and fig. 5.9(b), the query response time for the 250-GB and 1-TB datasets, i.e. AReM and Istabul, is shown, respectively. The query response time for STOS (exact kNN query processing) ranged from 166 ms to 1425 ms over AReM and from 773 ms to 4246 ms over Istanbul for different values of k. Again, the query response time did not show a significant change for the different sizes of the datasets, but the number of dimensions and the



(a) RMSE



(b) Average Distance (radius)



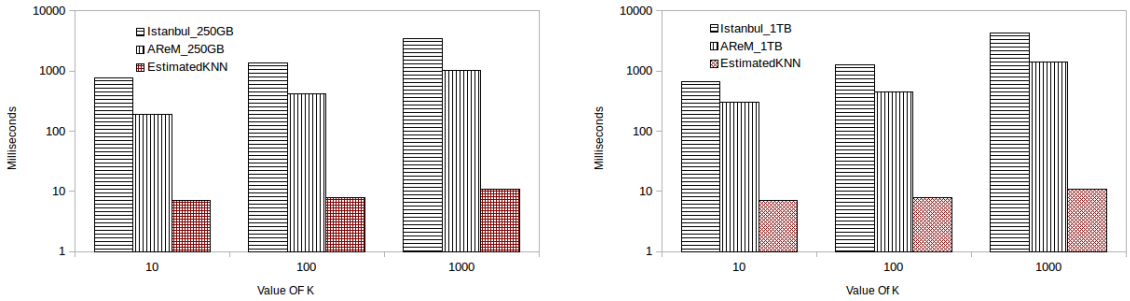
(c) Confidence Interval

Figure 5.8: Dataset Istanbul: Size 1 TB

different values of k significantly influenced the query response time, as expected. However, the query response time for the estimated kNN ranged from 7 ms to 12 ms for the different values of k. The query response time for the estimated kNN was not affected significantly by the number of dimensions. A small increase in the query response time was observed when the value of k increased. This implied that the estimated kNN query processing had up to three orders of magnitude lower query response time than STOS.

5.5.5 Prediction Assessment: Probabilistic kNN Regression Versus kNN Regression

In this section, the accuracy of the probabilistic kNN regression is extensively discussed by comparing the accuracy of probabilistic kNN regression with that of vanilla kNN regression. As previous experiments have already shown that the estimated kNN had several orders of magnitude lower query response time than the exact kNN, the main focus of the remaining experiments was to assess the accuracy of probabilistic kNN regression.



(a) Query Response Time For 250GB datasets (b) Query Response Time For 1-TB datasets

Figure 5.9: Query Response Time

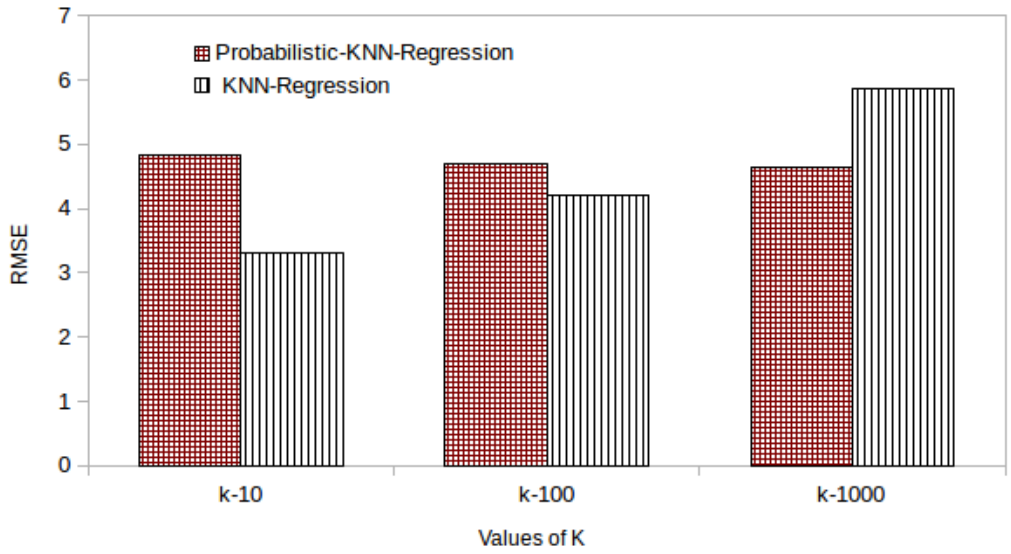


Figure 5.10: RMSE Dataset CCPP

RMSE, MAE, and coefficient of variation (CV) were used to measure the accuracy over three real-world datasets:

In Figure 5.10, the RMSE of probabilistic kNN and kNN is shown. The RMSE of kNN regression was lower than that of probabilistic kNN by 1.5 units when the value of k = 10 and 0.45 when k=100. However, when k=1000, probabilistic kNN had a higher accuracy than kNN regression by 1.2; this was expected because for k=1000, the vanilla kNN regression accessed 10% of the entire dataset. Moreover, note that the accuracy of the probabilistic kNN regression improved as the value of k increased. This was because as more plausible data points were generated to predict the value of the target RV, the variance of the generated points decreased. For instance, Figure 5.11 shows the mean absolute error (MAE) for both the methods; thus, the proposed method had 1.30 unit and 0.4 unit higher MAE for k=10 and k=100, respectively, but for k=1000, the proposed method had 0.93 unit lower MAE. However, as shown in Figure 5.12, the probabilistic kNN method had a lower coefficient of

variation (CV), particularly when k=100 and $k = 1000$. This implied that kNN regression had greater variability with respect to its MAE than the probabilistic kNN.

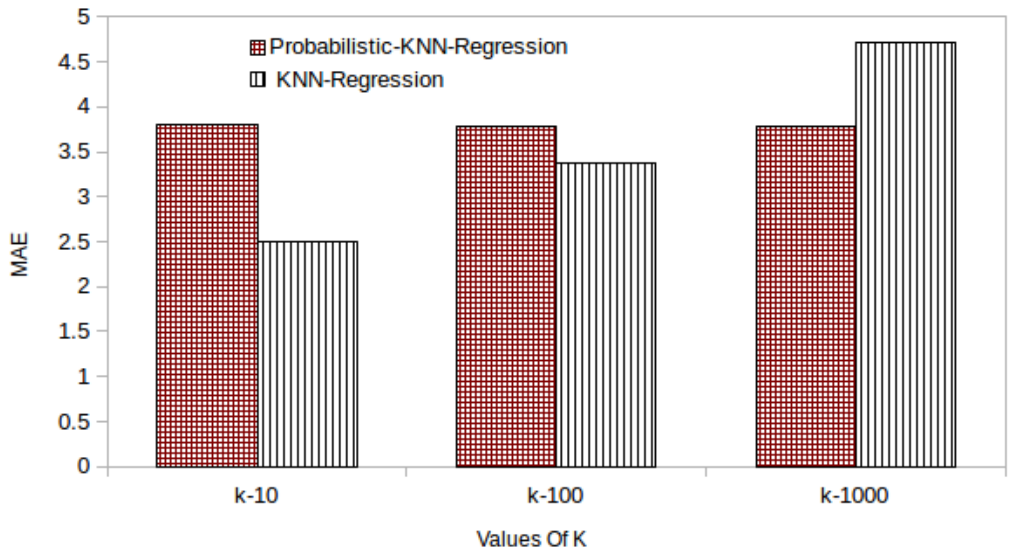


Figure 5.11: MAE Dataset CCPP

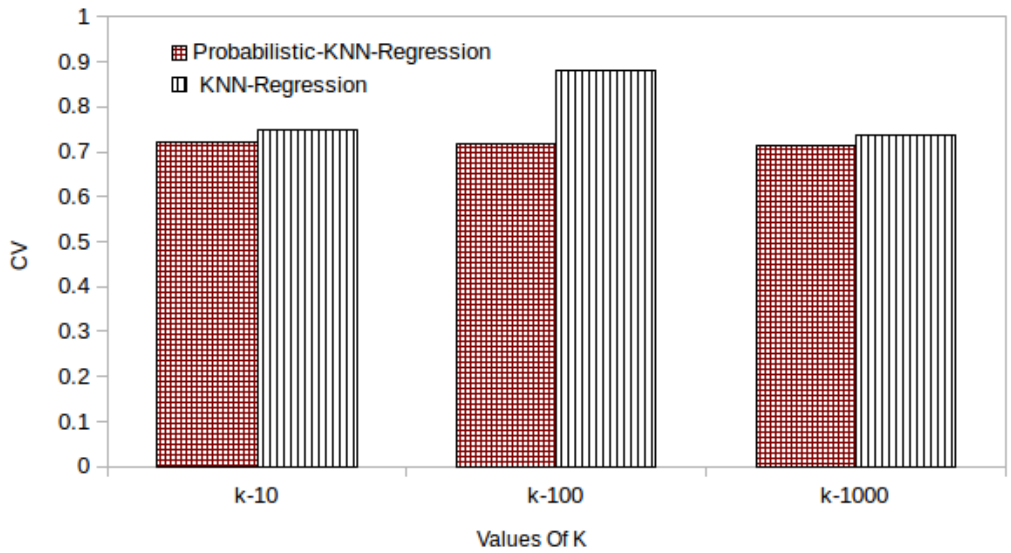
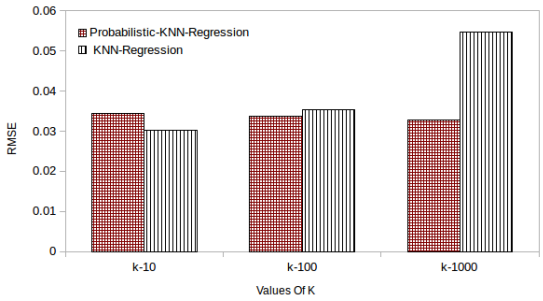
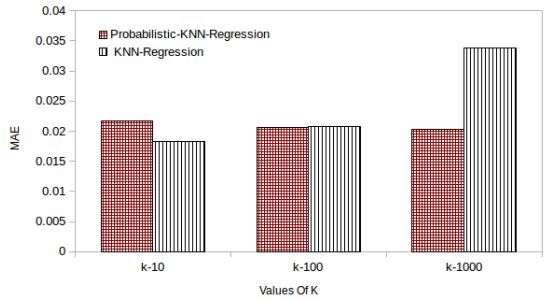


Figure 5.12: CV Dataset CCPP

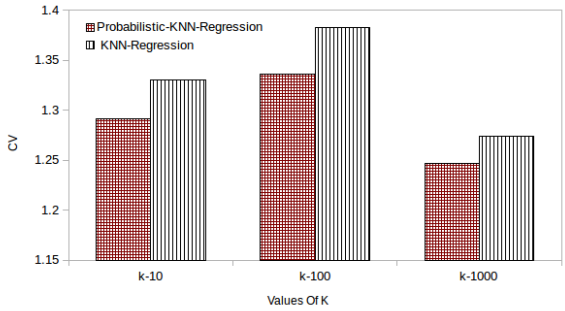
The same observation was noted for the other two datasets, as shown in figures 5.13(a), 5.13(b), 5.13(c), 5.14(a), 5.14(b), and 5.14(c). When k=10, on average, the kNN regression had from 12% to 18% lower RMSE and MAE, but the proposed approach had up to 18% lower variability with respect to its MAE. In contrast, when the value of k was 100, mixed results were observed; for the Abalon dataset, the proposed method had 3% and 4% lower RMSE and CV, respectively. However, for the Air dataset, the new method had 50% and



(a) RMSE



(b) MAE



(c) CV

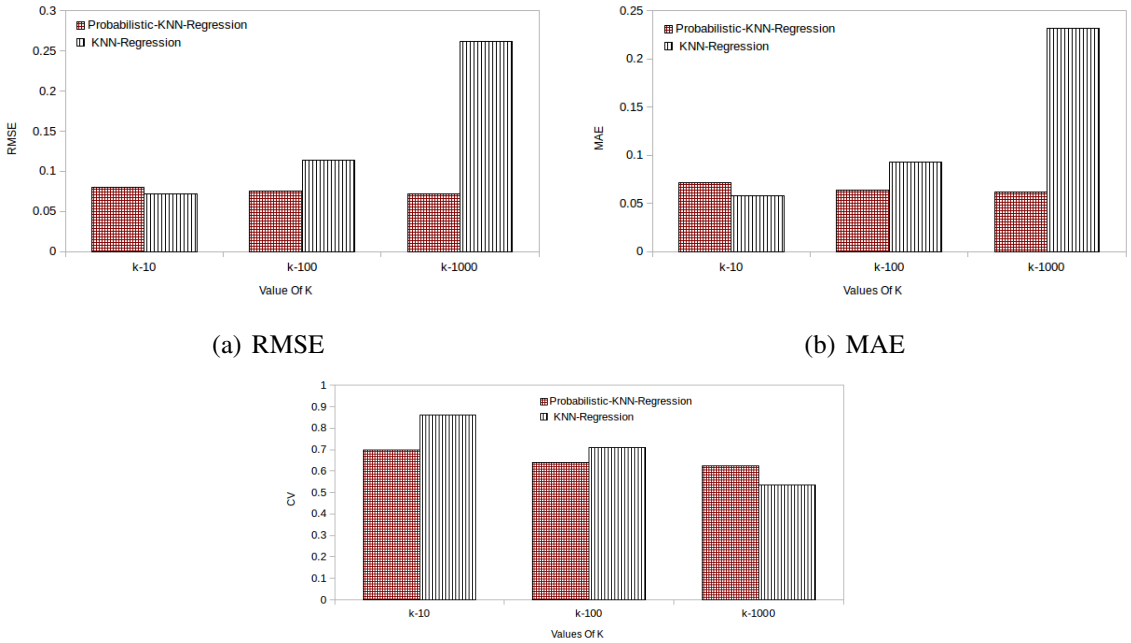
Figure 5.13: Abalon

1% lower RMSE and CV than the kNN regression, respectively. This demonstrated that the accuracy of the vanilla kNN regression deteriorated as the value of k and the number of dimensions of the dataset increased; however, the proposed method suffered less from the curse of dimensionality. Further research is required to assess the resilience of the proposed method against the curse of dimensionality.

5.6 Conclusions

In general, the *exact* kNN processing approaches have a high query response time for high-dimensional data. To tackle this problem, in the existing literature, approximated kNN processing approaches have been studied extensively. The core design philosophy of the approximated kNN query processing rests upon a solid design philosophy that states that the approximate nearest neighbour is almost as good as the exact one, particularly when the distance measure accurately captures the notion of user quality, small differences in the distance do not matter [9].

In this chapter, on the basis of such a fundamental tenet, a novel kNN processing approach is proposed. By capitalising the space transformation techniques explained in considerable depth in the previous chapter, the proposed approach called estimated kNN, unlike approximated kNN, estimates kNN to a query without accessing the underlying dataset. Moreover,



(a) RMSE

(b) MAE

(c) CV

Figure 5.14: Air Quality

the estimated kNN provides an average distance within which the actual kNN can be located with some degree of confidence or can receive the user-defined average distancea notion of user qualityand can notify the questionable user of the degree of confidence that the actual kNN can be located within the given distance from the estimated answers. The estimated kNN has the following attributes:

- high prediction accuracy,
- several orders of magnitude lower query response time, particularly for higher-dimensional data,
- no overhead cost of disk and network I/Os (dataless kNN processing),
- small memory footprint,
- flexibility to be deployed on the client side, thus reducing the workload at the coordinator (server), and
- flexibility to work in conjunction with STOSexact kNN processing method proposed in the previous chapterwithout any additional overhead cost.

In a similar vein, based on the space transformation technique, another novel method, probabilistic kNN regression, was proposed in this chapter. Probabilistic kNN regression estimates the k plausible values of the target RV on the basis of the predictor RVs of the given query.

Probabilistic kNN regression has all the advantages of the estimated kNN. Moreover, probabilistic k regression has a better tolerance to the curse of dimensionality than the vanilla kNN regression.

Chapter 6

The Pythia Framework

6.1 Introduction

In the previous chapters, several approaches were proposed for scaling exact kNN queries that run over low-dimensional large-scale datasets. The main factor for scaling exact kNN queries is accessing small but relevant subsets of a large-scale dataset. Hence, a dataset must be partitioned (indexed) in advance to query processing; but query response time and memory space requirement of the popular tree-based multi-dimensional indexing methods grows exponentially with dimension. In contrast, even though STOS, introduced in this thesis, successfully removes the dependency of memory space requirement on the dimension, query response time of STOS increases with dimension as well. For high-dimensional domain space, estimated kNN, introduced in the previous chapter, might be used to produce estimated kNN. However, when estimated kNN are of interest, all the other proposed methods are not suitable for high dimensional datasets.

Tackle the scaling-out problem of kNN in high dimensional space, a different way of indexing a high-dimensional large-scale dataset is proposed by the research group [8], and the proposed framework is called Pythia. Pythia was initially designed for the missing value imputation problem but can easily be integrated to be part of a highly scalable kNN framework that operates in Big Data environment. However, it worth to note that Pythia does not guarantee to retrieve exact kNN. Before explaining the contribution of this chapter, it is important to introduce Pythia in more detail.

Pythia A Missing Value Framework

In Big Data processing and Knowledge Management systems, data quality related issues pose significant challenges that potentially hinder data-processing algorithms. Arguably,

a missing values (MVs) problem is one of the most frequently occurring data quality related issues. For instance, MVs occur in results from medical experimentation and chemical analysis, in meteorology datasets, and microarray gene monitoring technology [25], in survey databases [12], in wireless sensors, and in survey questions when participants skip questions. Also, industrial and research databases include MVs [38], e.g., maintenance databases have up to 50% of their entries missing [53]. Patient records in medical databases lack some values; interestingly, a database of patients with cystic fibrosis missing more than 60% of its entries was analysed in [52].

Although MVs occur frequently, most of the on-shelf available data-processing algorithms, such as neural networks and support vector machines, either do not work well or fail in the presence of MVs; so such data-processing algorithms cannot be used for decision-making purposes [75]. Therefore, it is crucial to handle MVs appropriately in order to induce unbiased knowledge from a dataset.

Traditionally, before data-processing, MVs were either ignored, excluded or filled-in (*imputed*): imputation entails a *MV substitution algorithm* (MVA) that replaces MVs in a dataset with some plausible values. However, excluding data elements that contain MVs might lead to vital information loss, and hence imputing MVs is important as the imputed data can be treated as reliable as the observed data. Nevertheless, it is worth noting that the imputed values are as good estimations as the assumptions used to create them. Consequently, choosing the right MVAs improves the reliability of the imputed values.

The missing value problem in Big data environment

In a big data environment imputing missing data is extremely difficult due to the following three reasons [8]. (R1) The available missing value imputing algorithms (MVAs) can impute MVs with high accuracy. However, the missing value computation time of such algorithms depends on the size of a dataset. Thus, for a large-scale dataset imputation time can be significantly high. (R2) Due to the three Vs of a big data (volume, versatility and velocity), the existing MVAs cannot pace with the ever-growing, quickly changing, and speedily arriving data. Due to this issues, most MVAs in the literature are typically tested over small-to-medium size datasets. (R3) As the data user community is growing steadily, the request rate for MV imputation is high. These are considerable challenges for MV imputation in the big data environment.

For efficiently processing big data, Google researchers proposed MapReduce for distributed parallel processing of big data [30] that resides in a distributed file system such as the Google File System [42]. The central design philosophy of MapReduce is (i) distribute a dataset across several inexpensive machines and (ii) access the dataset distributed across those ma-

chines in parallel. MR has been solving different big data analytics related problem successfully.

However, vanilla MR might have high unnecessary overhead cost when executing several MVA such as kNN. For example, when kNN is used as MVA, vanilla MR accesses the entire dataset and such waste resources as explained in depth in the previous chapters; or when a MVA Expectation-Maximization (EM) is used, several expensive MR jobs (accesses to the entire dataset) are needed to compute a final answer [26]. During MVs imputation process, accessing the whole dataset that resides in all machines does not increase the accuracy of the estimated values but it might hurt the accuracy; thus accessing only a fraction of the machines that contain relevant data has the following significant advantages:

- *on MV imputation time*:as MV imputation time depends on the worst-performing machine, accessing all machines does not reduce the workload of the worse-performing machine but rather forces the imputation process to wait for the worst-performing machine to finish.
- *on accuracy of MVA*: as will be explained later, engaging all machines may actually introduce large additional MV estimation errors.
- *on imputation throughput of MVA*: if only a few machines are engaged for imputation, the rest of the machines instead of remaining ideal, can impute MVs of another data elements; ensuring inter-query parallelism.

To this end, Pythia was proposed to avoid accessing all machine during MV imputation. In Pythia, a dataset is randomly stored across different machines referred to as *cohorts*. During MV imputation, a cohort runs a MVA over a dataset stored locally. As MV imputation time depends on the size of a dataset, it decreases as more cohorts added to the framework. For example, assuming a dataset of size n is equally distributed across ten cohorts, and each cohort can run a MVA with asymptotic complexity $O(n^2)$ (or $O(n^3)$. MV imputation time of each cohort is expected to speedup input processing by a factor of $10^2 = 100$ (or $10^3 = 1000$); thus MVA runs on a dataset of size $\frac{1}{10}n$. Moreover, this alternative affords the possibility of accessing only a subset of all cohorts for MVs of an input vector.

In Pythia, an input vector is expected to contain some MVs in certain dimensions, and neither the input vector nor do the dimensions that contain MVs are known in advance. Once an input vector arrives in Pythia, MVs of the input vector must be estimated and imputed before storing the input vector.

The formidable challenges here include as follows: (i) estimation-error (accuracy) reasons, the estimation errors computed based on the subset of cohorts should be similar, if not better, compared to the errors produced by accessing the whole dataset; (ii) *swiftly* determine cohort/cohorts to engage per imputation, achieving large efficiency / scalability gains.

In the experiments, the performance (accuracy and imputation time) of Pythia is compared to a single machine that stores the whole dataset; hereinafter, this single machine is referred to as *Godzilla*. Godzilla can execute any MVA to impute MVs but might not able to cope with the ever-growing volume and velocity of a dataset.

Each cohorts in Pythia uses Adaptive Resonance Theory [20] (ART) to quantise dataset that stores locally. The general process consists the following steps: (1) each cohort clusters data, stored locally, using ART and send the cluster heads as a data digest (aka signatures) to a central machine called Pythia. (2) By exploiting the signatures received from all cohorts, Pythia maintains global knowledge regarding the distribution of a dataset across the cohorts. When an input vector with MVs arrives at the system, using the signature, Pythia predicts the appropriate cohort that contains relevant data to the impute vector. Then, Pythia sends the input vector to the selected (appropriate) cohorts only. Each of the selected cohorts independently execute an MVA - for example, kNN - and estimates values of the MVs; afterwards, Pythia receives estimated values from the selected cohort and computes the final estimated values by aggregating the estimates received from the cohorts.

Therefore, in this thesis, the performance of kNN in Pythia, in high dimensional data space, attracts interest. Considering that Pythia does not guarantee to retrieve exact kNN, but experimental results show that the estimation error of Pythia is comparable to that of exact kNN [8].

As Pythia utterly depends on signatures created by ART to identify relevant cohorts (which is very important for the accuracy of a kNN based estimation), in this thesis, the performance of Pythia (w.r.t estimation accuracy and imputation time) is investigated by adopting signatures created by a different clustering algorithm: the Self-Organising Maps [51](SOM). As we shall see later, Pythia performs almost the same with ART and SOM based signatures; thus, in this thesis, we found that Pythia is a robust system whose performance is independent of the clustering algorithms. Furthermore, the estimation accuracy of kNN in high dimensional space is compared to a more sophisticated MVA: the Expectation-Maximization [68] (EM). Even though kNN gains slightly better accuracy than EM, in general, both MVA have comparable accuracy.

However, even when accessing only one cohort, Pythia accesses the whole dataset that resides in the cohort. As nowadays a typical disk size is hundredth of Gigabytes if not Terabytes, accessing such huge dataset, even in parallel, might take considerable time and resources particularly when processing large-scale datasets. To alleviate such a problem, as we shall see later in the experimental results, more cohorts can be added to Pythia; but keep adding cohorts with the ever-growing dataset is not economically viable.

To this end, instead of accessing the entire dataset that resides in a cohort, *only relevant clusters that reside in appropriate cohorts can and should be accessed*. As we shall see later,

such method has comparable accuracy to that of the original design of Pythia but has lower imputation time; moreover, the imputation time of the proposed method does not significantly grow with a size of a dataset that resides in a cohort or with the number of data nodes in Pythia.

Last but not least, in case of high dimensional data, previous study [17] noted that the relative contrast of the distance of an input vector $\mathbf{i}$ with another vector $\mathbf{c}$ depends heavily on the adopted L_f distance metric where $L_f = \left(\sum_{i=1}^{d} (\mathbf{i}_i - \mathbf{c}_i)^f\right)^{1/f}$; this provides considerable evidence that the meaningfulness of the L_f worsens faster with increasing dimensionality for higher values of f. To this end, in order to study how Pythia (kNN) can be affected by L_f, two distance metrics, the Euclidean distance vs Manhattan distance, are compared. As we shall see later, even though no significant difference is noted, Manhattan distance has slightly better accuracy in high dimensional space.

6.2 Contribution

Originally the authors of Pythia [8] claim the following contributions:

- Pythia was the first study on scaling-out MV imputations.
- Pythia accesses relevant data partitions for missing value imputation and hence avoids accessing the whole dataset.
- for large-scale datasets, Pythia achieves a significant performance speed-up of MV imputation process comparing to MVA that runs in a single machine.
- Pythia has a better or comparable errors comparing to errors of MVA that runs on a single machine.

The main objective of this chapter is to answer the following three essential questions:

1. Is Pythia robust? As Pythia entirely depends on the signature (created by ART) to identify the relevant machine, investigate the performance -estimation accuracy and imputation time- of Pythia when it uses two different signatures constructed by two different clustering algorithms.
2. Is Pythia independent of MVA? Study the scalability of two MVA -kNN and the Expectation Maximization(EM) imputation [68]- w.r.t estimation accuracy and imputation time of Pythia.
3. Can Pyhtia access considerably smaller data elements when computing MVs? If so, how does it affect the scalability of the kNN ?

4. Does the accuracy of kNN as MVA affected when different distance metrics are used?

Consequently, the main findings are:

- When Pythia uses different clustering (signature building) algorithm, manages to produce almost identical imputation accuracy.
- Pythia scales well under both MVA; estimation accuracy of both methods does not show a large difference to that of Godzilla, and imputation time of MVA decreases as the size of data that processed by the algorithm decreases.
- Irrespective of the high dimensions, kNN based MVA of Pythia has comparable estimation errors in relation to the more sophisticated imputation algorithm and exact kNN.
- Instead of accessing the entire dataset that resides in relevant cohorts, accessing relevant clusters that reside in the relevant cohorts is possible.
- Accessing smaller data has comparable accuracy and significantly lower imputation time. Furthermore, the estimation accuracy and imputation time do not significantly affected by the number of cohorts.
- Even though, further studies are needed, Manhattan distance might have better applicability in Pyhtia when kNN is used as MVA in higher dimensional data space.

Part of this chapter was published in the Big Data journal as cited below:

- Cahsai, A., Anagnostopoulos, C. and Triantafillou, P. (2015) *Scalable data quality for big data: the Pythia framework for handling missing values*. Big Data, 3(3), pp. 159–172.

The rest of this chapter is organised as follows: in Section 6.3 most some MVA are discussed while in Section 6.4 definitions and more sophisticated MVA that are added in this work are presented. Section 6.5 reports on the original, and new clustering algorithms of Pythia are explained, and Section 6.6 provides a comprehensive performance evaluation of Pythia. Finally, Section 6.7 concludes the chapter with future directions.

6.3 Background and Related Work

6.3.1 Missing Value Imputation Problem & Algorithms

Assume $d \times n$ matrix $\mathcal{D}$ that contains n d-dimensional data points and another $d \times n$ indicator matrix $\mathcal{M}$ that records MV entries in $\mathcal{D}$; that is, if a value at $\mathcal{D}_{ij}$ is missing, then the value of

$\mathcal{M}_{ij}$ is 1; otherwise, $\mathcal{M}_{ij}$ is 0. The MVs are supposed to be modelled by a set of probability distribution $\mathcal{F}(\mathcal{M}|\theta)$, where θ denotes certain statistical parameters, and on the basis of the nature of this statistical distribution, MVs are classified as follows [57]:

Missing completely at random (MCAR)

If the probability of an observation being missing is independent of both the observed and the missing variables, it is called Missing Completely At Random (MCAR) and can be denoted mathematically as follows: :

$$P\{M = m|D, \theta\} = P\{M = m|\theta\} \text{ for all m, } \theta. \tag{6.1}$$

When MVs are MCAR, a statistical analysis of the complete data objects (ignoring data objects with MV) produces an unbiased result. In other words, in MCAR, any data element that contains MVs can be excluded from the dataset, and yet unbiased knowledge can be extracted from the remaining complete dataset.

Missing at random (MAR)

When the probability of the MVs is dependent on the observed data (variables) but is independent of the missing variable itself, the missed values are said to be MAR and can be denoted mathematically as follows:

$$P\{M = m|D, \theta\} = P\{M = m|D_{obs}, \theta\} \text{ for all m, } \theta. \tag{6.2}$$

Unlike MCAR, in this case, deleting data elements that contain missing at random introduces a bias into a dataset; thus, the knowledge extracted from such biased data might be misleading. MVs that are missing at random must be imputed on the basis of the complete data of the observed variables, before the execution of a data analytics algorithm.

Missing Not at Random (MNAR)

MVs related to the values of unobserved data are called MNAR. In other words, the probability of an MV depends on the missing variable itself. When neither MCAR nor MAR holds, then the data are MNAR. MNAR cannot be estimated using the existing variables and is rarely applicable in practice.

Case Deletion

One method of dealing with MVs is to delete all the data objects that contain the MVs [57]. If the data are MCAR, then a list-wise deletion yields an unbiased inference about the target population without any significant loss of information. However, if the case is not MCAR, then such a deletion will introduce a bias as one or more groups might be removed from the dataset.

Simple Mean Imputation

Perhaps, the simplest and the most used imputation method is the simple mean imputation. MVs are imputed by the mean, $\bar{x}$, of column i, in which the MVs reside in the data matrix $\mathcal{D}$. This mean can be denoted mathematically as follows:

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n}\mathcal{D}_{ji} \tag{6.3}$$

where i = column of the MV , j = the jth row, n = total number of rows, and $\mathcal{D}_{ij} \neq \emptyset$.

The variance underestimated by simple mean imputation measures how far on average the data objects in the given column deviate from their mean. Moreover, the mean is susceptible to the presence of outliers in the dataset. A few extreme observations affect the value of the mean to a considerable extent. Therefore, in most cases, simple mean imputation produces a poor estimated outcome of the MV.

Weighted K-nearest neighbors imputation

KNN is widely used in [4] because of its many advantages such as the following: it is a non-parametric method, which does not require the creation of a predictive model for each dimension with MVs and takes into account the correlation structure of the data. KNN is based on the assumption that the points close in terms of their distances are potentially similar. For given input $(\mathbf{x}_i, \mathbf{w}_i)$ with $\mathbf{x}_i = (\mathbf{z}_i, \mathbf{z}_i^{\mathbf{m}})$, KNN calculates a weighted Euclidean distance D_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j \in \mathcal{D}$ such that

$$D_{ij} = \left(\frac{\sum_{k=1}^{d} w_{ik}w_{jk}(x_{ik} - x_{jk})^2}{\sum_{k=1}^{d} w_{ik}w_{jk}}\right)^{1/2}.$$

The MV of the k-th dimension of $\mathbf{x}_i$ (i.e., $z_{ik}^{\mathbf{m}}$ of $\mathbf{z}_i^{\mathbf{m}}$) is estimated by the weighted average of non-MVs of the K most similar $\mathbf{x}_j$ to $\mathbf{x}_i$, i.e., $\hat{z}_{ik}^{\mathbf{m}} = \sum_{j=1}^{K} \frac{D_{ij}^{-1}}{\sum_{v=1}^{K} D_{iv}^{-1}} x_{jk}$. KNN is typically

used with K=10,15,20; theses values have been favored in previous studies [4], [50]. (In this chapter k=10 was used when imputing MVs).

Remark 1. *A naive approach for searching the closest d-dimensional data point with respect to a given point* $\mathbf{x}$ *over a dataset of size $n = |\mathcal{D}|$ requires $O(nd)$ time. This implies $O(ndK)$ time for retrieving the K nearest data points. Nonetheless, one could build a d-dimensional tree over the points of $\mathcal{D}$ to allow one to efficiently perform the K nearest neighbour search. Such a structure is good for searches in low-dimensional spaces. However, its efficiency decreases with an increase in the dimensionality, and in high-dimensional spaces, this structure provides no performance benefits over a naive $O(ndK)$ linear search.*

Expectation maximisation imputation

The EM algorithm is an iterative algorithm for estimating MVs by maximising the likelihood function [68]. Assume that $\mathcal{D}$ is generated by a probability density function $f(\mathcal{D}|\theta)$, where θ is a parameter of the model. The likelihood function $L(\theta|\mathcal{D})$ is a function of the parameter θ for fixed $\mathcal{D}$. For mathematical convenience, the likelihood function is represented by its log-likelihood function $l(\theta|\mathcal{D}) = \ln(L(\theta|\mathcal{D}))$. Without any loss of generality, consider for fixed $\mathcal{D}$, a set of parameters $\theta = \{\theta_1, \ldots, \theta_t\}$ with $t > 0$. For every $\theta_i \in \theta$, $l(\theta_{i \leq t}|\mathcal{D})$ is calculated. The obtained outcome shows how likely $\mathcal{D}$ is observed under θ_i. The highest the outcome is, the most likely it is that $\mathcal{D}$ is observed under this parameter. In general, L is used to identify the value of θ, which is best supported by $\mathcal{D}$. For a set $\mathcal{D}$, which contains observed values, $\mathcal{D}_{obs}$, and MVs $\mathcal{D}_{miss}$, the log maximum likelihood of $\mathcal{D}$ is $l(\theta|\mathcal{D})$ = $l(\theta|\mathcal{D}_{obs}, \mathcal{D}_{miss})$ = $l(\theta|\mathcal{D}_{obs})$ + $\ln f(\mathcal{D}_{miss}|\mathcal{D}_{obs}, \theta)$. The main concept of EM is to maximise the maximum likelihood estimation (MLE) of θ from $l(\theta|\mathcal{D}_{obs})$ in order to maximise the MLE of $l(\theta|\mathcal{D})$. The EM algorithm consists of the following four steps: Step (i) Replace MVs with the estimated values. Step (ii) Estimate θ (also known as E-step). Step (iii) Re-estimate MVs using the new θ (referred to as M-step). Step (iv) Re-estimate θ, and iterate until convergence. [68].

To illustrate how the EM algorithm is used for imputation, consider the d-dimensional mean vector $\mathbf{u} = [u_1, \ldots, u_d]^\top$, and covariance matrix $\Sigma = [\sigma_{jk}]$ with $j, k = 1, \ldots, d$. Both $\mathbf{u}$ and Σ refer to the learning parameter θ; i.e., $\theta = (\mathbf{u}, \Sigma)$. Initially, μ and Σ are calculated considering only the non-missing values, i.e. from the $\mathcal{D}_{obs}$ set. Then, the EM imputation algorithm calculates each step as follows:

- Step 1: For each $\mathbf{x}_i \in \mathcal{D}$, if $w_{ik} = 0$ then one has to estimate $\hat{z}_{ik}^{\mathbf{m}} = u_k$. Note that $\mathbf{w}_i$ remains unchanged in order to help him identify which dimensions are observed or missed in the original dataset $\mathcal{D}$.

- Step 2: Estimate parameter θ_t at iteration $t \geq 1$. For each $k, j = 1, \ldots, d$ one has to calculate the following:

$$E\left(\sum_{i=1}^{|\mathcal{D}|} x_{ik} | \mathcal{D}_{obs}, \theta_t\right) = \sum_{i=1}^{|\mathcal{D}|} x_{ik}^t$$

and

$$E\left(\sum_{i=1}^{|\mathcal{D}|} x_{ik} x_{ij} | \mathcal{D}_{obs}, \theta_t\right) = \sum_{i=1}^{|\mathcal{D}|} x_{ik}^t x_{ij}^t + c_{jki}^t$$

with

$$\hat{z}_{ik}^{\mathbf{m}} = \begin{cases} x_{ik}, & \text{if } w_{ik} = 1 \\ E\left(\sum_{i=1}^{n} x_{ik} | \mathcal{D}_{obs}, \theta_t\right) & \text{if } w_{ik} = 0 \end{cases}$$

and

$$c_{jki}^t = \begin{cases} 0, & \text{if } w_{ik} = 1 \text{ or } w_{ij} = 1 \\ x_{ik} x_{ij} | \mathcal{D}_{obs}, \theta_t & \text{if } w_{ik} = 0 \text{ and } w_{ij} = 0 \end{cases}$$

At the end of this step, the purpose is to estimate the sufficient statistics, i.e. mean, variance, and covariance, so that the following step can update the parameter θ_t. In particular, this step estimates $\mathbf{u}$ and Σ and uses them to build a set of regression equations that predict the missing values from the $\mathcal{D}_{miss}$ set. This is achieved by the *sweep* regression operator [68] realising the conditional expectations. Such an operator combines the mean vector and the covariance matrix into a single augmented matrix and applies a series of transformations that produce the desired regression coefficients and residual variances.

- Step 3: Re-estimate MVs by using the new θ_t parameter. This step becomes a straightforward estimation problem that uses the filled-in sufficient statistics from the previous step to impute the missing values. Then, for each $k, j = 1, \ldots, d$, calculate the following:

$$u_k^{(t+1)} = (|\mathcal{D}| - 1)^{-1} \sum_{i=1}^{|\mathcal{D}|} \hat{z}_{ik}^{\mathbf{m}}$$

and

$$\sigma_{jk}^{(t+1)} = (|\mathcal{D}| - 1)^{-1} \sum_{i=1}^{|\mathcal{D}|} [(x_{ij} - \mu_j)(x_{ik} - \mu_k) + c_{jki}]$$

- Step 4: If $|l(\theta_{t+1}|\mathcal{D}) - l(\theta_t|\mathcal{D})| \leq \epsilon$, then terminate (converge); otherwise, go to Step 2. ϵ is set to $\epsilon = 10^{-3}$ for the convergence.

Remark 2. *Each iteration takes $O(nd)$ computations given that $n = |\mathcal{D}|$. However, the termination behaviour of EM is not easy and not guaranteed. Theoretically speaking, without*

any stopping threshold (or setting a stopping threshold $\epsilon = 0$), EM would infinitely converge to an infinite precision, i.e. $\epsilon = 0$. Hence, the theoretical runtime of EM is infinite. Any small and non-negative threshold $\epsilon > 0$ and $\epsilon \to 0$ forces EM to terminate earlier. However, it will be difficult to obtain a theoretical limit here that is different from $O(ndt_\epsilon)$, where t_ϵ is the number of iterations required for achieving a precision close to ϵ.

6.3.2 Other missing value imputation algorithms

The MV imputation algorithm proposed by [49] fills an MV on the basis of the estimated distribution of the dataset. A regression-based imputation was proposed by [68]: assuming dimensions have a relationship among each other, MVs are imputed by the regression of the corresponding dimensions. Another likelihood-based imputation was proposed by [68]; this method estimates the parameter of a dataset by using the maximum likelihood or maximum a posterior procedures relying on the variants of the expectation-maximisation (EM) algorithm. The multiple imputation algorithm was proposed by [70], wherein each MV is replaced by a set of plausible values that represent the uncertainty regarding the actual missing value. These multiple-imputed datasets are then analysed using the standard procedures for complete data, and the results from these analyses are combined. Based on dynamic Bayesian networks, MV imputation for time series was proposed by [55]. Other MV imputation approaches were proposed in [89, 64]; the first approach uses matrix completion, while the second approach uses Gaussian mixture clustering for imputing MVs. Moreover, machine learning methods such as decision trees and rule-based methods have been used to impute MVs [37]. Finally, the imputation framework [38] applies most of the existing MVAs to improve their accuracy of imputation. An interested reader can refer to [38], [56], and [4] for a comprehensive survey of the most recent MVAs.

6.4 Definitions

6.4.1 Notations

Definition 12. *[8] A dataset is represented by a set $\mathcal{D}$ such that $\mathcal{D} = \{\mathbf{d}_1, \ldots, \mathbf{d}_{|\mathcal{D}|}\}$. For each vector $\mathbf{d}_i \in \mathcal{D}$, there exists a corresponding vector $\mathbf{w}_i$ such that $\mathbf{w}_i = [w_{ik}]^\top$ with $w_{ik} = 0$ whenever $\mathbf{d}_i$'s k-th dimensional value is missing; otherwise, $w_{ik} = 1$. Moreover, $\mathbf{x}_i$ is expressed as $(\mathbf{z}_i, \mathbf{z}_i^{\mathbf{m}})$, where $\mathbf{z}_i \in \mathbb{R}^{d'}$ denotes the observed values and $\mathbf{z}_i^{\mathbf{m}} \in \mathbb{R}^{(d-d')}$ denotes MVs, with $d' = \sum_{k=1}^{d} w_{ik}$.*

Definition 13. *[8] $\mathcal{D}$ is divided into y finite partitions such that $y > 0$ and $\mathcal{D} \equiv \cup_{i=1}^{y} \mathcal{D}_i$ and $\mathcal{D}_i \neq \mathcal{D}_j$ when $i \neq j$. The ith machine (cohort) is dented as S_i; thus, $\mathcal{S} = \cup_{i=1}^{y} S_i$ is the set*

of all cohorts. S_i maintains $\mathcal{D}_i$ locally and executes an MVA over $\mathcal{D}_i$; in contrast, a single machine (Godzilla) is denoted as S_0 and runs an MVA over $\mathcal{D}$.

Definition 14. *[8] A single MV input on MVA is $\mathbf{i} = (\mathbf{d}, \mathbf{w})$, and the output is $\hat{\mathbf{x}}$ expressed by $(\mathbf{z}, \hat{\mathbf{z}}^{\mathbf{m}})$. $\hat{\mathbf{x}} \in \mathbb{R}^d$ is referred to as estimate containing $\hat{\mathbf{z}}^{\mathbf{m}} \in \mathbb{R}^{(d-d')}$ of imputed MVs by the MVA. If $\mathbf{x}_a$ is the actual vector, the absolute reconstruction error can be expressed as follows: $e = \| \hat{\mathbf{d}} - \mathbf{d}_a \|$; $\| \mathbf{d} \|$ denotes the Euclidean norm.*

Definition 15. *[8] $\mathcal{D}$ is divided into y finite partitions such that $y > 0$ and $\mathcal{D} \equiv \cup_{i=1}^{y} \mathcal{D}_i$ and $\mathcal{D}_i \neq \mathcal{D}_j$ when $i \neq j$. The ith machine (cohort) is dented as S_i; thus, $\mathcal{S} = \cup_{i=1}^{y} S_i$ is the set of all cohorts. S_i maintains $\mathcal{D}_i$ locally and executes a MVA over $\mathcal{D}_i$; in contrast, a single machine (Godzilla) is denoted as S_0 and runs a MVA over $\mathcal{D}$.*

Definition 16. *[8] A single MV input on MVA is $\mathbf{i} = (\mathbf{d}, \mathbf{w})$ and output is $\hat{\mathbf{x}}$ expressed by $(\mathbf{z}, \hat{\mathbf{z}}^{\mathbf{m}})$. $\hat{\mathbf{x}} \in \mathbb{R}^d$ is referred to as estimate containing $\hat{\mathbf{z}}^{\mathbf{m}} \in \mathbb{R}^{(d-d')}$ of imputed MVs by MVA. If $\mathbf{x}_a$ is the actual vector, the absolute reconstruction error is $e = \| \hat{\mathbf{d}} - \mathbf{d}_a \|$; $\| \mathbf{d} \|$ denotes the Euclidean norm.*

6.4.2 Missing value algorithms in the Pythia framework

The original authors of Pythia [8] used the weighted K-nearest neighbours (KNN) [77] and REG [68] MVAs. However, as mentioned earlier, when the weighted K-nearest neighbours (KNN) [77] are used, Pythia does not guarantee the identification of the exact kNN when imputing MVs. In this thesis, as kNN was the main topic of interest, the quality (w.r.t imputation time and particularly MVs estimation accuracy) of the kNN, retrieved from a selected cohort (this will be explained later in detail), was compared with that of another more sophisticated MVA, the expectation maximisation (EM) [70], computed by the same cohort. To this end, EM was implemented as an MVA in Pythia.

6.5 The Pythia Framework Functionality

Pythia uses several cohorts for MV imputation. Each cohort,$\mathcal{S}_i$, locally maintains a subset $\mathcal{D}_i$ of a potentially large dataset such that $\mathcal{D}_i \in \mathcal{D}$. A cohort can invoke an MVA over the data partition stored locally and send the estimated imputed values to a central machine, the Pythia.

When an input d-dimensional vector that contains the MVs arrives at Pythia, Pythia swiftly predicts the appropriate cohort or cohorts, $\mathcal{S}'$, in which the MVA is going to be executed. Note that Pythia predicts the relevant cohorts, $\mathcal{S}' \subseteq \mathcal{S}$, using a set of signatures (data digest) created by each cohort and sent to Pythia to be used as an index [8], i.e. Pyhtia maintains a

global signature, $\mathcal{P} = \{P_i\}_{i=1}^{y}$; thus, Pythia has the global knowledge of how $\mathcal{D}$ is distributed across the cohorts.

The idea behind a signature is that P_i (a signature) can sufficiently represent the data elements stored in the ith cohort, S_i; thus, when an input vector with a missing value arrives at Pythia, on the basis of these signatures, the most relevant cohorts to the input vector are identified, and the input vector is then sent only to the selected cohorts to carry out the missing value imputation process. The operation of the framework is as follows: Given **i**,

- Step 1: Pythia predicts $\mathcal{S}' \subseteq \mathcal{S}$ with respect to $\mathcal{P}$.
- Step 2: Pythia engages only the cohorts from S' sending the input **i** to them.
- Step 3: Each cohort $S_i \in \mathcal{S}'$
  - Step 3.1: S_i invokes an MVA and
  - Step 3.2: S_i provides its estimate $\hat{\mathbf{d}}_i$ to Pythia.
- Step 4: Pythia constructs the aggregate estimate $\hat{\mathbf{d}}$ that is sent to the cohorts from $\mathcal{S}'$.
- Step 5: Each $S_i \in \mathcal{S}'$ can exploit $\hat{\mathbf{d}}$ for updating its P_i.
- Step 6: Pythia uses $\hat{\mathbf{d}}$ for updating $\mathcal{P}$.

6.5.1 Signatures

In a lower-dimensional space, how the multidimensional index approaches, such as QT and STOS, are used for indexing a dataset stored in a distributed system is explained in detail in the previous chapters. However, as discussed earlier, such indexing approaches have no better performance gain than that of a linear kNN search in a high-dimensional space. In an effort to tackle such an issue, the authors of Pythia proposed an unsupervised competitive learning-based algorithm for the quantisation of the domain space, $\mathcal{D}_i$, and then use the cluster heads aka signature, P_i, to represent $\mathcal{D}_i$; thus, P_i can serve as an index to avoid a linear scan by identifying a relevant subset of $\mathcal{D}$, over which an MVA such as kNN runs.

In the original paper, the adaptive resonance theory (ART) [20] was used for constructing signatures. In this chapter, another adaptive vector quantisation algorithm, self-organising maps (SOM) [51], is added, and the performances of the two algorithms are studied w.r.t identifying the most relevant cohort by quantifying the accuracy of the estimated MVs.

ART signature

ART [20], whose algorithm is shown in Algorithm 3, is a member of the unsupervised learning model from the competitive learning paradigm. ART constructs P_i by clustering the data elements of $\mathcal{D}_i$. During the clustering processes, a data element $\mathbf{d}_k \in \mathcal{D}_i$ is assigned either to an existing cluster or to a new cluster on the basis of the following two criteria. (Criteria 1) If the distance between $\mathbf{c}^*$, the closest cluster head, and $\mathbf{d}_k$ is less than some pre-defined radius (*vigilance*), i.e. $\parallel \mathbf{c}^* - \mathbf{d}_k \parallel \leq \rho_i$ for some *vigilance* $\rho_i > 0$, then $\mathbf{d}_k$ is assigned to the closest cluster. Then, the cluster head, $\mathbf{c}^*$, is updated as follows: $\mathbf{c}^* \leftarrow \mathbf{c}^* + \eta_i(\mathbf{d}_k - \mathbf{c}^*)$, where $\eta_i \in (0, 1)$ is the learning rate, which decreases gradually. (Criteria 2) If the distance between $\mathbf{c}^*$, the closest cluster head, and $\mathbf{d}_k$ is greater than the (*vigilance*), i.e. $\parallel \mathbf{c}^* - \mathbf{d}_k \parallel > \rho_i$, then $\mathbf{d}_k$ is assigned to a new cluster; the head of the new cluster is set to $\mathbf{d}_k$, i.e. $\mathbf{c} = \mathbf{d}_k$ and $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\mathbf{c}\}$. Keep in mind that the closest cluster to data element $\mathbf{d}_k$ is defined as follows: $\mathbf{c}^* = \arg\min_{\mathbf{c} \in \mathcal{C}_i} \parallel \mathbf{c} - \mathbf{d}_k \parallel$, where $\mathcal{C}_i$ is the set of cluster heads.

Definition 17. *[8] The ART signature P_i of cohort S_i over $\mathcal{D}_i$ is the triple*

$$P_i = \langle \mathcal{C}_i, \rho_i, \eta_i \rangle. \tag{6.4}$$

Algorithm 3: ART-based Signature Algorithm at Cohort S_i

Input: $\mathcal{D}_i, \eta_i, \rho_i$
Output: $\mathcal{C}_i$
$\mathcal{C}_i = \{\mathbf{d}_1\}$;
for $1 < k \leq |\mathcal{D}_i|$ **do**
  $b^* = \parallel \mathbf{c}^* - \mathbf{d}_k \parallel = \min_{\mathbf{c} \in \mathcal{C}_i} \parallel \mathbf{c} - \mathbf{d}_k \parallel$; **if** $b^* > \rho_i$ **then**
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\mathbf{d}_k\}$;
  else
    $\mathbf{c}^* \leftarrow \mathbf{c}^* + \eta_i(\mathbf{d}_k - \mathbf{c}^*)$;
  end
end

Definition 18. *[8] $\boldsymbol{d}$ can be said a member of an ART P_i signature, denoted by $\boldsymbol{d} \in P_i$, iff $\min_{\mathbf{c} \in \mathcal{C}_i} \parallel \mathbf{c}' - \boldsymbol{z}_i \parallel \leq \rho_i$; otherwise, $\boldsymbol{d} \notin P_i$, where $\boldsymbol{z}_i$ is the observed values of $\boldsymbol{d}$ and $\mathbf{c}'$ is the corresponding values of* $\mathbf{c}$.

When a new data element, say $\mathbf{d}$, that contains MVs arrives at Pythia, in order for $\mathbf{d}$ to be assigned to a cohort, $\mathcal{S}_i$, the following criterion must be satisfied: $\parallel \mathbf{c}' - \mathbf{z}_i \parallel \leq \rho_i$, where $\mathbf{c} \in \mathcal{C}_i$, $\mathbf{z}_i$ is the observed values of $\mathbf{d}$ and $\mathbf{c}'$ contains corresponding values of $\mathbf{c}$, i.e values of $\mathbf{c}$ that correspond to the observed dimensions of $\mathbf{d}$. The fundamental concept of such a design is that when $\mathbf{d} \in P_i$ (i.e. when the above criterion is satisfied), $\mathcal{S}_i$ contains data elements similar to $\mathbf{d}$. Thus, when imputing MVs, $\mathcal{S}_i$ can produce a more accurate estimation than the

other cohorts whose representatives do not satisfy the criterion. To this end, the larger the number of representatives $\mathbf{c} \in \mathcal{C}_i$ that satisfy the criterion is, the more appropriate is $\mathcal{D}_i$ for $\mathbf{d}$.

As ρ_i represents a threshold of similarity between points and representatives, it guides the ART algorithm in determining when a new representative should be formed. In order to give a physical meaning to ρ_i, it is expressed through a set of percentages $\alpha_k \in (0, 1)$ of the ranges between the lowest $x_k^{\min}$ and the highest $x_k^{\max}$ values of each dimension k of points in $\mathcal{D}_i$, $k = 1, \ldots, d$. Let $\mathbf{r}_i = [(x_1^{\max} - x_1^{\min}), \ldots, (x_d^{\max} - x_d^{\min})]^\top$ and the diagonal $d \times d$ matrix $\mathbf{A}$ with $\mathbf{A}[k, k] = \alpha_k$. Then, $\rho_i = \| \mathbf{A}\mathbf{r}_i \|$. High α_k values result in a small number of representatives and vice versa. Each S_i determines a ρ_i over $\mathcal{D}_i$, creates P_i through Algorithm 3, and sends P_i to Pythia.

SOM Signature

A self-organising map (SOM) [51], whose algorithm is shown in Algorithm 4, is an unsupervised learning model and is formally a non-linear, ordered, smooth mapping of similar high-dimensional vectors to the nearby lattices $\mathcal{L}$ in a two-dimensional space. SOM implicitly captures the structure of a high-dimensional dataset and identifies the representatives of the dataset that have similar statistical characteristics; in essence, SOM produces a discretised representation of a dataset by mapping similar vectors in the domain space to the neighbouring representatives; thus, the representations conserve the underlying distribution of the dataset $\mathcal{D}$.

Similar to the case of ART, when SOM is used to build P_i, cohort S_i produces representatives of the dataset $\mathcal{D}_i$ stored locally. For each data element such that $\mathbf{d} \in \mathcal{D}_i$, the online SOM algorithm incrementally maps $\mathbf{d}$ into a lattice $\mathcal{L}$ composed of $\ell_i \times \ell_i$ representatives, $\ell_i > 0$; i.e., the number of representatives in S_i is ℓ_i^2. Hereinafter, the parameter ℓ is referred to as the lattice width.

The representatives are linked together by a neighbourhood relationship $h(j, j')$ over the indices $j, j' \in \mathcal{L}$, $j, j' = 1, \ldots, \ell_i^2$. With each representative on $\mathcal{L}$, a representative vector $\mathbf{c}_j$ is attached, and $\mathbf{c}_j$ has the same dimension as the underlying dataset. Each representative, $\mathbf{c}_j$, is initialised by drawing a vector from $\mathcal{D}_i$ at random. Thereafter, for every $\mathbf{d} \in \mathcal{D}_i$, the winner lattice is identified by the winning representative $\mathbf{c}_j^*$ that matches best with $\mathbf{d}$; then, $\mathbf{d}$ is assigned to be a member of the winner lattice. As shown in equation 6.5, the best match is defined in terms of the shortest distance between $\mathbf{c}_j$ and $\mathbf{d}$.

$$j^* = \arg\min_{j \in \mathcal{L}} \mathcal{E}(\mathbf{d}, \mathbf{c}_j). \tag{6.5}$$

In the Euclidean space where $\mathcal{E}(\mathbf{d}, \mathbf{c}_j) = \| \mathbf{d} - \mathbf{c}_j \|_2$, i.e. the 2-norm, the online SOM algorithm, at the kth input $\mathbf{d}_k$, $k = 1, \ldots, |\mathcal{D}_i|$, consists of the following two steps:

- Step 1: (Assignment) Vector $\mathbf{d}_k$ is assigned to a winning representative $\mathbf{c}_j^*$; i.e. $\| \mathbf{d}_k - \mathbf{c}_j^* \| = \min_{j \in \mathcal{L}} \| \mathbf{d}_k - \mathbf{c}_j \|$
- Step 2: (Update) All the representatives in $\mathcal{L}$ are updated as follows:

$$\mathbf{c}_j = \mathbf{c}_j + \eta(k) h(j, j^*; k) \left(\mathbf{d}_k - \mathbf{c}_j \right). \tag{6.6}$$

The parameter $\eta(k) \in (0, 1)$ called the learning rate is a non-increasing function of k. A good choice of $\eta(k)$ significantly improves the convergence of SOM [51]; usually, $\eta(k) = \frac{\eta(k-1)}{1+\eta(k-1)}$ with $\eta(0) = 1$. A discussion about $\eta(k)$ and the choice of an optimal learning rate can be found in [51]. Next, $h(j, j^*; k)$ is a smoothing kernel function defined over indices $j, j^* \in \mathcal{L}$, usually given by the following Gaussian neighbourhood function:

$$h(j, j^*; k) = \exp\left(-\frac{\| \mathbf{r}_j - \mathbf{r}_j^* \|^2}{2\beta^2(k)} \right).$$

Vectors $\mathbf{r}_j$ and $\mathbf{r}_j^*$ are, respectively, the locations of representatives $\mathbf{c}_j$ and $\mathbf{c}_j^*$ on $\mathcal{L}$. The topological neighbourhood is symmetric around the winning representative, which has the maximum value. Parameter $\beta(k)$ is the width of the neighbourhood with the initial value β_0 defined as $\beta(k) = \beta_0 \exp(-\frac{k}{T_\beta})$, where T_β is a constant. The boundaries of neighbourhood $h(j, j^*; k)$ depends on $\beta(k)$. A small width value corresponds to narrow boundaries, while with a high width, the boundaries contain more neighbours.

Remark 3. *As the size of $|\mathcal{D}_i| = \frac{1}{m}|\mathcal{D}|$ is expected to be large, it is safe to assume that the online SOM algorithm converges. In a case where the algorithm converges early, before scanning the entire dataset, the updating of the representatives can be stopped when a termination criterion is satisfied. This criterion, $\epsilon > 0$, refers to the 1-norm between successive estimates of the representatives, and the algorithm converges when $\sum_{j \in \mathcal{L}} \|\mathbf{c}_j(k) - \mathbf{c}_j(k-1)\|_1 < \epsilon \cdot \sum_{j \in \mathcal{L}} \|\mathbf{c}_j(k-1)\|_1$ with $\|\mathbf{c}_j\|_1 = \sum_i^d |c_{ji}|$ and d $\mathbf{c}_j(k)$ is the location of $\mathbf{c}_j$ when processing the kth data element of $\mathcal{D}_i$ and $k > 0$.*

Let $\mathcal{C}_i$ be the set of representatives $\{\mathbf{c}_j\}_{j=1}^{\ell_i^2}$ belonging to lattice $\mathcal{L}$.

Definition 19. *The SOM signature P_i of cohort S_i over $\mathcal{D}_i$ is the tuple*

$$P_i = \langle \mathcal{C}_i, \ell_i \rangle. \tag{6.7}$$

Each cohort, $S_i \in \mathcal{S}$, can independently set the number of representatives ℓ_i^2; thus, each cohort has the flexibility to determine the resolution (quality) of the data space quantisation. Intuitively, the higher the value of ℓ_i is, the more fine-grained is the resolution of the $\mathcal{D}_i$ quantisation. However, a high value of ℓ_i requires a relatively large memory space in Pythia. In contrast, a low value of ℓ_i might not be sufficient to represent the diversity of the data in $\mathcal{D}_i$.

When an input vector $\mathbf{d}$ that contains MVs arrives at Pythia, the distance between the input vector $\mathbf{d}$ and $\mathbf{c}_j$, $\mathbf{c}_j \in \mathbf{P}$, is computed on the basis of the observed values of $\mathbf{d}$. After identifying the closest representative (winner representative), a cohort, $\mathcal{S}_i$, represented by the winner representative is selected as the appropriate machine to be engaged on the imputation process of $\mathbf{d}$.

However, the notion the closest representative only indicates that across all the representatives stored in Pythia, $\mathbf{c}_{j^*}$ is the closest to $\mathbf{d}$ but does not provide information regarding the other cohorts that might contain data elements similar to $\mathbf{d}$. Recall that in the case of ART, the following criterion is used to identify cohorts that contain similar data elements: $\| \mathbf{c} - \mathbf{d} \| \leq \rho_i$; see definition 18. To this end, in case of SOM, to identify cohorts that contain data elements similar to those of $\mathbf{d}$, a smooth distance metric between $\mathbf{d}$ and $\mathbf{c}_j$, a *degree of membership* of $\mathbf{d}$ to P_i, is defined by the function $\mu_i : \mathbb{R}^d \rightarrow [0, 1]$ such that

$$\mu_i(\mathbf{d}) = \exp(- \| \mathbf{d} - \mathbf{c}_{j^*} \|_2^2). \tag{6.8}$$

A μ_i value close to unity indicates that $\mathbf{d}$ is topologically very close to a representative $\mathbf{c}_j$; thus, $\mathbf{d}$ is believed to be a member of P_i with a high degree. A μ_i value close to zero indicates that $\mathbf{d}$ is topologically very distant from a representative $\mathbf{c}_j$; therefore, $\mathbf{d}$ is not a member of P_i.

With the use of a user-defined *degree of membership*,ϵ, all the cohorts whose representatives satisfy $\epsilon < \mu_i$ can be selected as relevant machines to impute the MVs of $\mathbf{d}$. The larger the number of representatives $\mathbf{c} \in \mathcal{C}_i$ that satisfy the criterion is, the more appropriate is $\mathcal{S}_i$ to process $\mathbf{d}$.

After the prediction of the relevant cohorts, the input vector $\mathbf{d}$ is sent only to the selected cohorts, each of which, in turn, estimates the MVs of $\mathbf{d}$ independently and sends back the estimated values to Pythia. Finally, Pythia computes the (final) estimate $\hat{\mathbf{d}}$ by aggregating all the estimated values received from the selected cohorts.

Definition 20. *One can say that* $\boldsymbol{d}$ *is a member of an SOM* P_i*, denoted by* $\boldsymbol{d} \in_\mu P_i$*, with a degree of* $\mu_i(\boldsymbol{d}) > \epsilon$, $\epsilon \rightarrow 0$.

Algorithm 4: SOM-based statistical signature algorithm at cohort S_i

Input: $\mathcal{D}_i$, ℓ_i, β_0, T_β
Output: $\mathcal{C}_i$
Initialize $\mathbf{c}_j$, $j \in \mathcal{L}$;
$\mathcal{C}_i = \{\mathbf{c}_1, \ldots, \mathbf{c}_{\ell_i^2}\}$;
for $(1 \leq k \leq |\mathcal{D}_i|)$ **do**
  $j^* = \arg\min_{j \in \mathcal{L}} \parallel \mathbf{d}_k - \mathbf{c}_j \parallel_2$;
  $\mathbf{c}_j \leftarrow \mathbf{c}_j + \eta(k)h(j, j^*; k)(\mathbf{d}_k - \mathbf{c}_j)$, $j \in \mathcal{L}$;
end

Remark 4. *Once Pythia produces the estimate* $\hat{\boldsymbol{d}}$ *given an input* $\mathbf{i} = ((\mathbf{z}, \hat{\mathbf{z}}^{\mathbf{m}}), \mathbf{w})$*, it updates locally the signatures of the cohorts which were engaged in the imputation process. In the case of ART signatures, the reader could refer to [8] which reports on the expected magnitude of the change in the representatives in an ART signature because of the estimate. In the case of SOM signatures, the updates are the same as those of the ART signature, provided that the winner representative* $\mathbf{c}_{j^*}$ *of input* $\mathbf{z}$ *gets updated with a small constant rate* η*; the same rate is adopted in ART signatures.*

6.5.2 Cohort prediction process

Thus far, we have seen that with the use of the signatures stored in Pythia, relevant cohorts are identified in order to participate in the imputation process of the MVs for an input vector d. In the case of ART, all the cohorts whose representatives satisfy definition 18 are selected to participate in the imputation processes; similarly, in SOM, the relevant cohorts are selected on the basis of definition 20.

The focus of this section is two-fold: MV imputations must (i) be low cost and (ii) have high accuracy. MVs must be imputed quickly (low cost): the total MV imputation time consists of the elapsed time for (a) communication between Pythia and the cohorts and (b) executing MVA at the cohort. In contrast, high accuracy refers to low RMSE. In [8], the authors reported that Pythia has lower cost and higher accuracy than Godzilla.

Furthermore, in this work, SOM is introduced as a mechanism for building signatures in Pythia, and the performance of SOM, w.r.t the imputation time and accuracy, is compared to that of Godzilla and of ART-based Pythia. Both implementations of Pythia, SOM-based and ART-based, engage the top-$\mathcal{K}$ relevant cohorts for an imputation request, $1 \leq \mathcal{K} \leq m$. As Pythia communicates only with the relevant cohorts and the relevant cohorts, in turn, run the MVA in parallel, Pythia has low imputation cost and is expected to have comparable accuracy to that of Godzilla.

Top-1 cohort prediction

The authors of Pythia proposed two different schemas for predicting cohorts: (1) top-$\mathcal{K}$ cohort prediction and (2) top-1 cohort prediction. The first schema predicts the top-$\mathcal{K}$ cohorts; the second schema, however, engages only one cohort which runs the MVA over the dataset that it stores locally.

In particular, when processing kNN over a large-scale dataset, one of the main objectives of this thesis is to study the effect of accessing as few data elements as possible; thus, the top-1 cohort prediction schema resonates with the objectives of the thesis. Accordingly, in this chapter, only the top 1-cohort prediction schema is of interest, and the performance of the 1-cohort prediction, in both the ART and SOM settings, is compared to that of Godzilla.

Cohort prediction in ART

Given imputation request $\mathbf{i}$ and a set of ART signatures, Pythia determines the best cohort $S^* \in \mathcal{S}$ with $P^* = \langle \mathcal{C}^*, \rho^*, \eta^* \rangle$ such that the following criteria hold true:

- Criterion C1: $\mathbf{c}^* = \arg\min_{\mathbf{c} \in \cup_{i=1}^{m} \mathcal{C}_i} \parallel \mathbf{c} - \mathbf{z} \parallel$ and $\mathbf{c}^* = \arg\min_{\mathbf{c} \in \mathcal{C}^*} \parallel \mathbf{c} - \mathbf{z} \parallel$; i.e., $\mathbf{c}^* \in \mathcal{C}^*$ is the closest representative to $\mathbf{z}$ among all the representatives from all the signatures.
- Criterion C2: $\mathbf{z} \in P^*$; i.e., the vector $\mathbf{z}$ is a member of the ART signature P^*.

Note that the number of observed values d' of an input vector $\mathbf{d}$ is defined as $d' = \sum_{k=1}^{d} w_k < d$, where $0 < d'$ and is represented by a vector $\mathbf{z}$ such that $\mathbf{z} \in \mathbb{R}^{d'}$. Thus, the number of MVs in $\mathbf{d}$ is equal to $d - d'$. For determining whether $\mathbf{z} \in P^*$, Pythia computes $\rho^{*(d')} \leq \rho^*$; thus, the distance between the input vector $\mathbf{d}$ and a representative $\mathbf{c}$ is computed on the basis of the observed values of $\mathbf{d}$; hence, Pythia checks whether $\parallel \mathbf{c}^* - \mathbf{z} \parallel \leq \rho^{*(d')}$ holds when predicting the winner representative. If no cohort satisfies criteria C1 and C2, then Pythia engages the cohort that satisfies only criterion C1.

Cohort prediction in SOM

Consider the top-1 (best cohort) scheme. Given imputation request $\mathbf{i}$ and a set of SOM signatures, Pythia determines the best cohort $S^* \in \mathcal{S}$ as follows:

- Step 1: Find the winner prototype $\mathbf{c}_i^* = \arg\min_{\mathbf{c} \in \mathcal{C}_i} \parallel \mathbf{z} - \mathbf{c} \parallel$ from signature S_i, $\forall i$.
- Step 2: Define a membership indicator $I_i(\mathbf{z}) = 1$ if $\mu_i(\mathbf{z}) > \epsilon$; otherwise, 0 $\forall i$.

- Step 3: Calculate the normalised membership degree

$$\tilde{\mu}_i(\mathbf{z}) = \frac{\mu_i(\mathbf{z})I_i(\mathbf{z})}{\sum_{k=1}^{m} \mu_k(\mathbf{z})I_k(\mathbf{z})}$$

  and select the cohort with the maximum value of $\tilde{\mu}$.

If for all cohorts $S_i \in \mathcal{S}$, do not satisfy the membership criteria, i.e $I_i(\mathbf{z}) = 0$, $\mathbf{z}$ cannot be represented by any winner representative from all the signatures, then Pythia engages the cohort whose winner representative is the closest to input $\mathbf{z}$ among all the winner representatives (from all the signatures). If $\mathcal{K} > 1$, then Pythia engages (at most) the top $\mathcal{K}$ cohorts ranked with respect to the $\tilde{\mu}$ value. In this case, the final $\hat{\mathbf{d}}$ is produced by aggregating all $\hat{\mathbf{d}}_j$ with $1 \leq j \leq \mathcal{K}$.

6.5.3 On accessing only the relevant clusters

In the previous chapters, we demonstrated that accessing a small chunk of a dataset improves the query response time and the scalability of the kNN algorithm significantly. Also, the Pythia framework shows that the scalability of the kNN MVA improves when, as we shall see later, the number of cohorts increases.

However, compared to the methods proposed earlier in the thesis, Pythia accesses significantly large part of a dataset; however, recall that the methods proposed earlier are not suitable for high dimensional data and Pythia has a notable advantage as can process high dimensional data. Having said that, even in the case of the 1-top (best cohort) schema, Pythia accesses the entire dataset that resides in a cohort. As nowadays a typical disk size is hundredth of Gigabytes if not Terabytes, accessing such huge dataset, even in parallel, might take considerable time and resources particularly when processing large-scale datasets.

For that end, the following solution is proposed in this chpater: (step-1) cluster a dataset that resides in a cohort and store each cluster separately; (step-2) given imputation request $\mathbf{i}$ and a set of SOM or ART signatures, Pythia predicates all clusters whose representatives satisfies the following criterion: $\exp(- \parallel \mathbf{c_{ij}} - \mathbf{z} \parallel_2^2) > \epsilon$, where ϵ is users defined threshold, in the experiments discussed in this chapter ϵ was set to 0.8 - 0.9; and (step-3) after predicting the relevant clusters, the input vector is send to the cohorts that store the clusters; each of those cohorts, in turn, run kNN MVA over the selected clusters only, not on the entire dataset.

6.5.4 The effect of distance metrics on kNN in high dimensional space

In the case of high-dimensional data, for imputing MVs, Pythia (and also a cohort)has to compute the Euclidean distance between the observed dimensions of an input vector and a signature (or data element that reside in the cohort). In case of high dimensional data, the Euclidean distance metric might change in some non-obvious ways [3]; thus retrieving kNN becomes ill-defined. In such a scenario, the concept of proximity may not be meaningful from a qualitative perspective; hence, this could be a fundamental problem along with the performance degradation of high-dimensional algorithms.

Previous study [17] noted that the relative contrast of the distance of an input vector $\mathbf{i}$ with another vector $\mathbf{c}$ depends heavily on the adopted L_f distance metric where $L_f = \left(\sum_{i=1}^{d} (\mathbf{i}_i - \mathbf{c}_i)^f\right)^{1/f}$; this provides considerable evidence that the meaningfulness of the L_f worsens faster with increasing dimensionality for higher values of f. In this chapter, therefore, how Pythia would be affect by L_f, is studied by comparing the Euclidean distance vs Manhattan distance, when identifying relevant cohort and computing MVs using kNN.

6.6 Performance Evaluation

6.6.1 Experimental Setup

Using a real-world dataset, which is downloaded from the UCI Machine Learning Repository[33], an extensive series of experiments were conducted to evaluate the performance of Godzilla and Pythia- over the best cohort scheme ($\mathcal{K} = 1$). From the dataset, which is called physical activity monitoring features, 1.2 million real-valued 51-dimensional ($d = 51$) vectors that have no MVs were selected randomly; furthermore, from the dataset, which is called gas sensor array temperature modulation, 3.8 millions real-valued 20-dimensional ($d = 20$) vectors were used. Synthetically MVs were randomly introduced into the datasets and were independently marked as missing with probability $q \in (0, 1)$. Therefore, $|\mathcal{D}| \sum_{k=1}^{d-1} \binom{d}{k} q^k (1-q)^{d-k}$ points with MVs were expected for each datasets. To test the robustness of Pythia w.r.t accuracy, a relatively high probability of MVs per dimension, $q = 0.3$, was set; 0.3 is a random choice.

In SOM, the size of the lattices is varied $\ell \in \{10, 20, 30, 50\}$. As shown in Fig. 6.1, no significant change in the root mean square error (defined formally later) is observed when ℓ increases from 20 to 50. As no significant trade-off between accuracy and lattice size is noted, the lattice size is set to $\ell = 20$ to increase the efficiency of Pythia (w.r.t memory space requirement for storing signatures and winner cohort prediction time). When $\ell = 20$, each

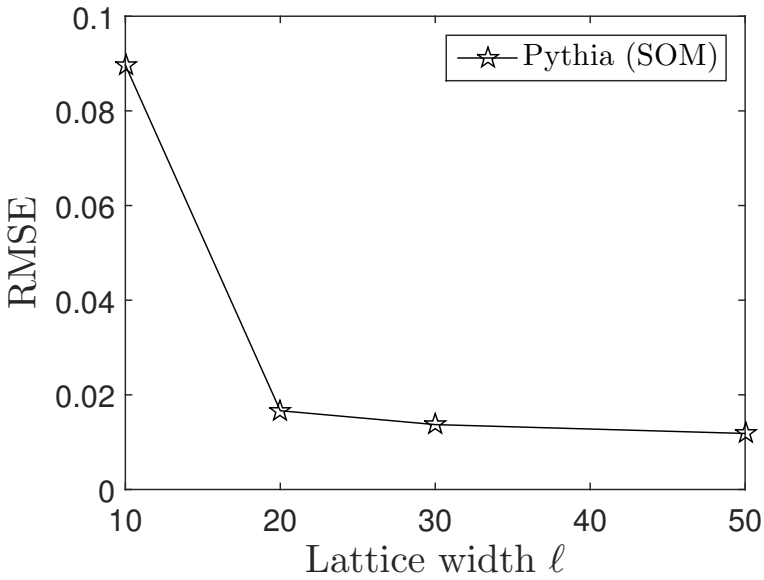


Figure 6.1: RMSE vs. lattice width ℓ for SOM signature.

SOM signature contains a very small fraction of points of the entire set $\mathcal{D}$. In contrast, for a fair comparison, in ART, on average signature P_i contains around 400 points. In both cases, a minute fraction of $\mathcal{D}$ is sent to Pythia from each cohort as representatives.

In SOM the initial width of the neighbourhood function boundaries $\beta_0 = \ell$ (the width of the lattice) is adjusted to contain almost all the representatives during the first few iterations and decays exponentially with each iteration.In SOM, an initial learning rate is set to $\eta(1) = 0.1$; as discussed in Section 6.5.1, the initial learning rate decreases gradually. In ART learning rate is set to $\eta = 0.1$. All experiments were run 1,000 times and took their average values for all performance metrics. The number of cohorts m ranges in $\{20, \ldots, 100\}$.

Pythia's cohorts prediction algorithms and the MVAs reported in Section 6.4.2 were developed in Java; for the EM MVA a Java library called weka [84] was used. Also, mean missing value imputation algorithm, see section 6.3.1, is used as a baseline and the performance, particularly accuracy of the estimated values of Pythia, is compared to that of mean estimation. Table 6.1 summarises the parameter values used in our experiments.

6.6.2 Performance Metrics

Two different metrics were considered: *efficiency metrics* and *accuracy metrics*. Scale-out systems, such as Pythia, that employ m machines (cohorts) can support two types of parallelism: *intra-parallelism* and *inter-parallelism* . The former refers to the capability of processing a single task (e.g. MVs imputation of a single vector) using several machines that run in parallel; whereas, the latter refers to the capability of processing several tasks (e.g.

Parameter	Notation	Value/Range
d	dimension	51 and 20
m	number of cohorts	$\{20, 40, 60, 80, 100\}$
q	MV probability	0.3
$\|\mathcal{D}\|$	dataset size	$1.2 \cdot 10^6$ and $3.8 \cdot 10^6$
T	imputation requests	10^3
α	vigilance range in ART for the 51-dim data	[0.38, 0.32, 0.28, 0.26, 0.23]
α	vigilance range in ART for the 20-dim data	[0.175,0.16,0.15, 0.14,0.13]
η	learning rate in ART	0.1
ℓ	lattice width in SOM	$\{10, 20, 30, 50\}$
β_0	initial width in SOM	20

Table 6.1: Experimental parameters.

MVs imputation of multiple vectors), each of which runs in parallel in a subset of machines. It is worth to note that Godzilla affords neither of these parallelisms. The second scenario is more efficient (quick and scalable) when a system is presented with a large batch of input. Given this, the efficiency metrics, which are used in this work, embody various efficiency aspects impacting scalability.

First, *imputation latency*- the time (in milliseconds) a system (i.e., Godzilla or Pythia) requires to impute a single input (vector) using a MVA- is reported. The rate of latency increase as dataset sizes grow is a strong aspect of scalability. In Pythia, latency refers to the time to predict best cohort S^*, plus the latency to run MVA in parallel at the engaged cohort.

Imputation speedup is defined as the ratio of Godzilla latency over Pythia latency; it indicates how much a system is faster than Godzilla for a single imputation. The linear imputation speedup ratio is m.

In a case of the mean MVA, as the mean of each dimension can be computed only once, then stored in memory, has a constant imputation time; so no need to report either imputation latency or imputation speedup of the mean MVA.

Imputation Accuracy is also measured using RMSE: the root mean squared difference between actual vector $\mathbf{D}_a$ and estimated vector $\hat{\mathbf{D}}$ after T imputation requests. RMSE is defined formally as follows:

$$RMSE = \left(\frac{1}{T} \sum_{t=1}^{T} \frac{\sum_{k=1}^{d} w_{tk} (x_{(a)tk} - \hat{D}_{tk})^2}{\sum_{k=1}^{d} w_{tk}} \right)^{1/2}. \tag{6.9}$$

Last but not least, the *storage* metric b for Pythia adopting either ART or SOM signatures is measured; *storage* metric quantifies a total number of representatives in the ART signature

and in the SOM signature. For SOM, b is defined by: $b = \ell^2 m$ assuming that all SOM signatures adopt the same lattice width ℓ. In case of ART signature, based on the underlying distribution of each cohort's datasets, the signature of a cohort is made up of a different number of representatives. If ξ_i is the number of representatives of an ART signature P_i then the storage metric corresponds to $b = \sum_{i=1}^{m} \xi_i$. Thus, in ART a cohort contains 400 signatures on average; in contrast, in SOM every cohort contains exactly 400 signatures.

6.6.3 Imputation Efficiency

For Pythia, with ART and SOM signatures the speed-up of the EM and the KNN imputation algorithms over a different number of cohorts m are shown, for the physical activity monitoring features dataset, in figures 6.2 and 6.3, respectively. In Figure 6.2, for the EM MVA, a slightly superlinear speedup was observed, particularly when the number of cohorts increased; for the kNN MVA in Figure 6.3 a superlinear speedup was observed for all the number of cohorts, i.e. the speedup ratio is a slightly greater than m when m cohorts are engaged in the imputation process. In contrast, with ART and SOM signatures, a slight supperlinear and liner speedup were noted, on the gas sensor array temperature modulation dataset, for the EM and kNN as shown in fig. 6.4 and fig. 6.5, respectively. This demonstrates that as a performance of MVA such as KNN and EM is significantly affected by the size of a dataset $\mathcal{D}$, i.e., their computational complexity is proportional to $O(|\mathcal{D}|)$, running a MVA over a subset of a dataset $|\mathcal{D}_i| = \frac{1}{m}|\mathcal{D}|$ that resides in a cohort S_i reduces the imputation latency by at least a factor of m. Furthermore, concerning computational effort, the more demanding an imputation algorithm is, the more speedup achieved when running it over a partition of the entire datasets. Overall, Pythia has gained considerable speedup achievement using ART and SOM signatures in the EM and the KNN MVA.

For Godzilla and Pythia (with ART and SOM signatures) for the 51-dimensional dataset, the latency in milliseconds is shown for EM and kNN in Figures 6.6 and 6.7, respectively, for a different number of cohorts m. The dataset size decreases as the number of cohort increases; thus, Pythia, with both signatures, scales well as its latency significantly decreased with the number of cohorts. Indicatively, when kNN was used, Godzilla requires 42 seconds and Pythia (with $m = 100$ and SOM signature) requires 0.3 seconds to impute MVs of an input vector. In contrast, the same observations were noted for the 20-dimensional dataset as show in fig. 6.8 and fig. 6.9 for EM and kNN, respectively. Thus by adding more cohorts, Pythia can effectively process a big data with missing values. Up to now, a strong case for the scale-out advantages of the Pythia framework is demonstrated by the experimental results shown so far. No significant difference in speed up is noted between the SOM and ART based implementations.

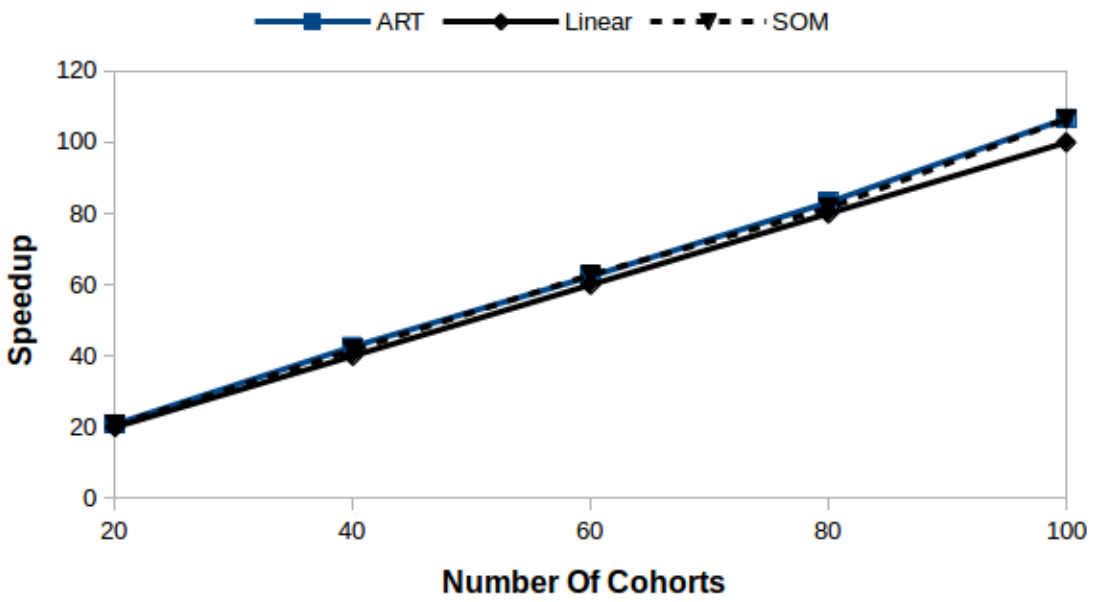


Figure 6.2: Speedup vs. number of cohorts m for ART and SOM signatures using EM. Dataset: Physical activity monitoring features .

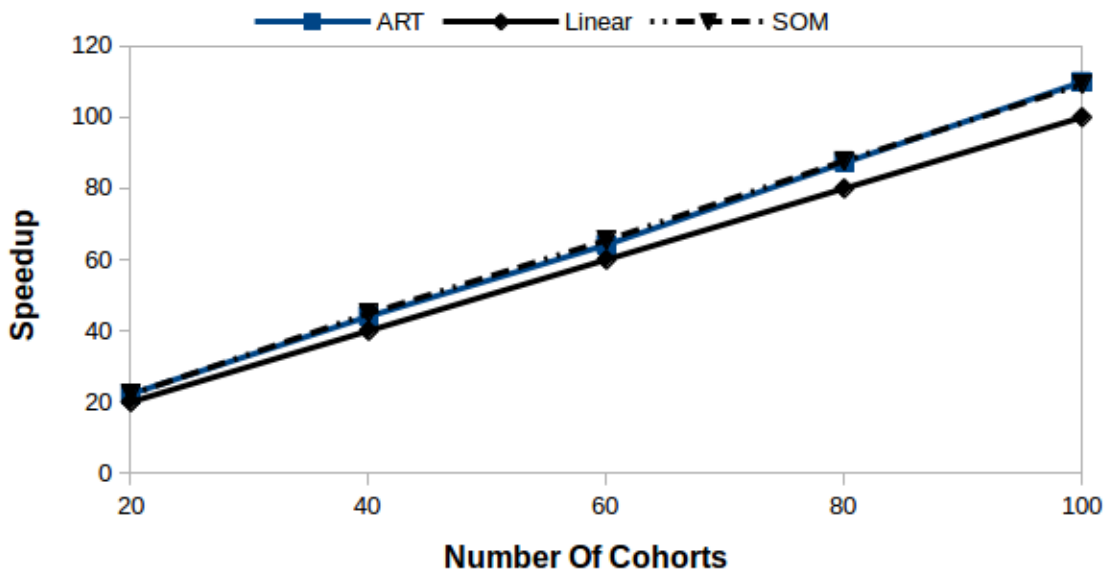


Figure 6.3: Speedup vs. number of cohorts m for ART and SOM signatures using KNN, $K = 10$. Dataset: Physical activity monitoring features.

6.6.4 Imputation Accuracy

Now let us focus on imputation accuracy. The expected imputation accuracy of Pythia, with both the ART and SOM based signatures, is computed on the basis of of the best cohort prediction scheme, which engages only the best cohort out of the m cohorts.

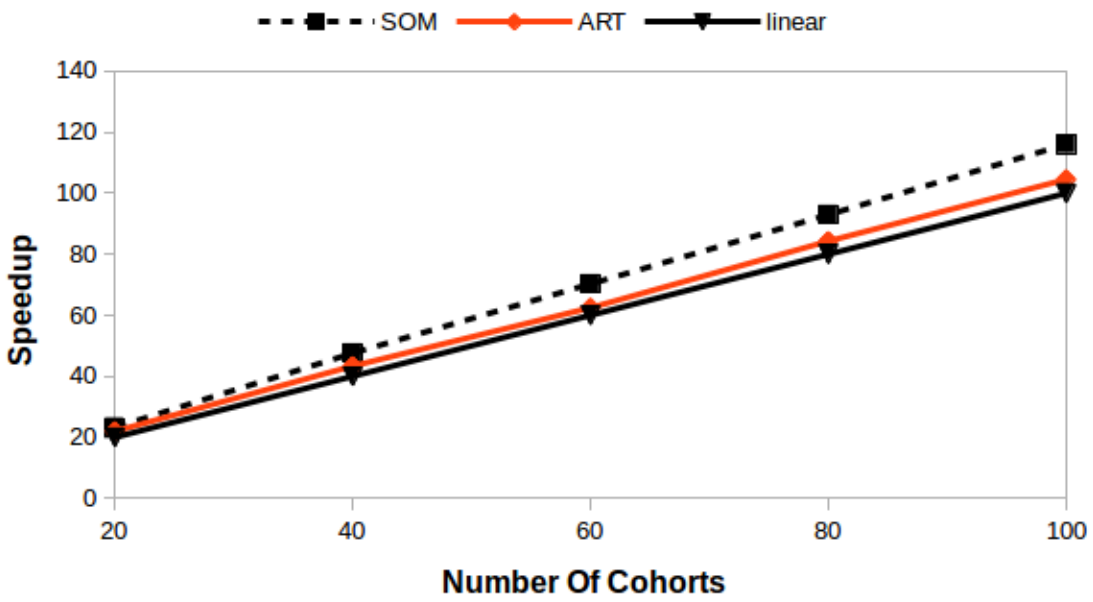


Figure 6.4: Speedup vs. number of cohorts m for ART and SOM signatures using EM. Dataset: Gas sensor array temperature modulation.

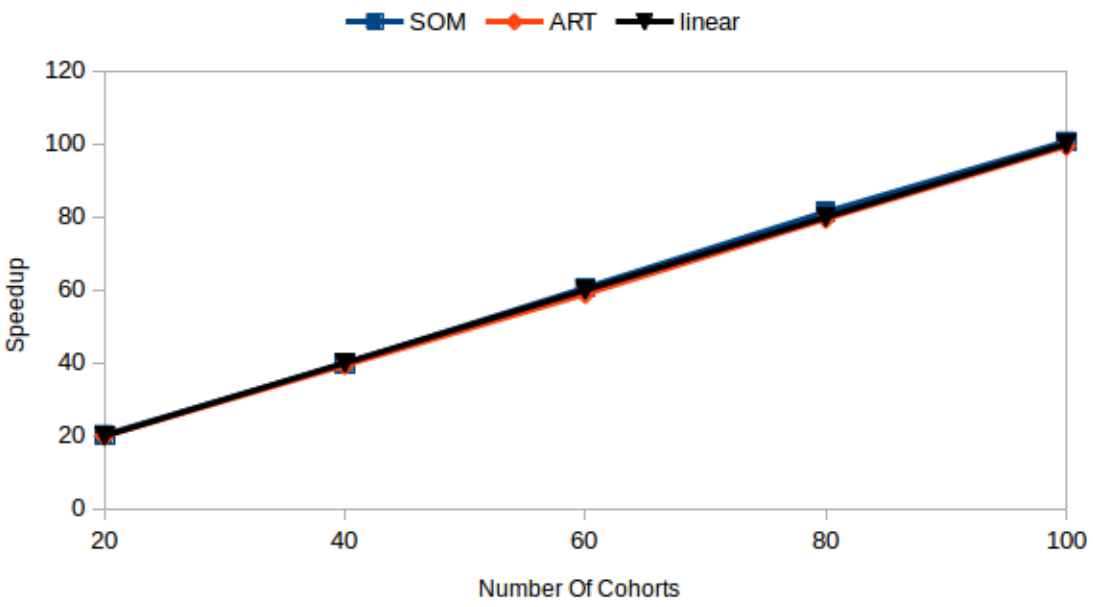


Figure 6.5: Speedup vs. number of cohorts m for ART and SOM signatures using KNN, $K = 10$. Dataset: Gas sensor array temperature modulation.

The RMSE for the KNN and the EM algorithms against a different number of cohorts m, for the 51-dimensional data, is shown in Figures 6.10 and 6.11, respectively. As shown in Figure 6.10, on average for all m, Pythia's KNN (in both ART and SOM signatures) had relatively low RMSE. The RMSE of the mean MVA was the highest as expected. Note that, in spite of Godzilla had slightly lower RMSE, see Figure 6.10, the RMSE difference to that of Pythia

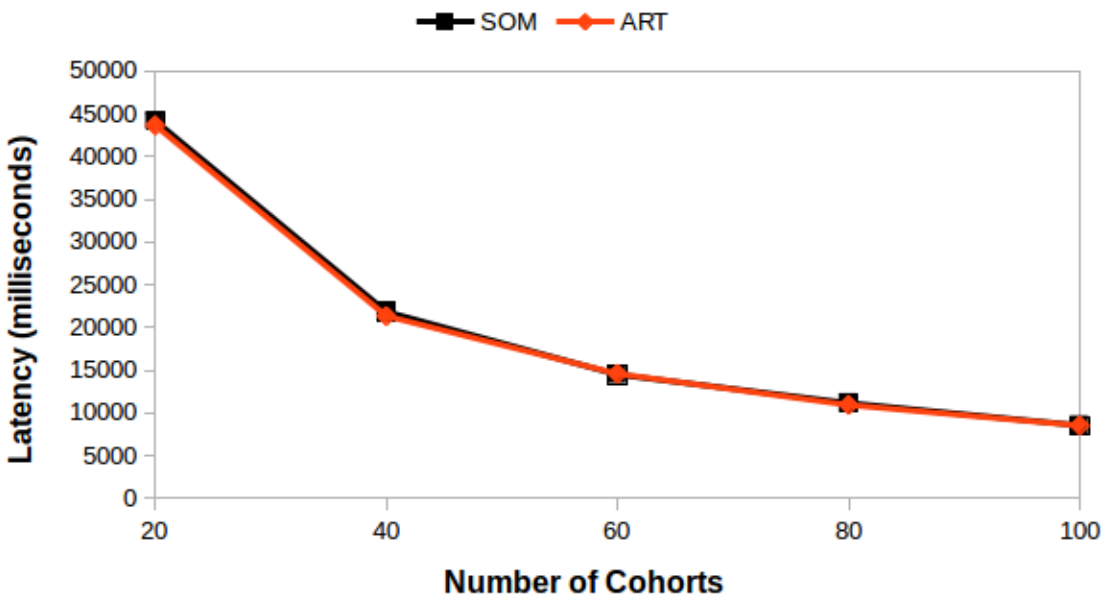


Figure 6.6: Latency in milliseconds vs. different number of cohorts based on EM. Dataset: Physical activity monitoring features.

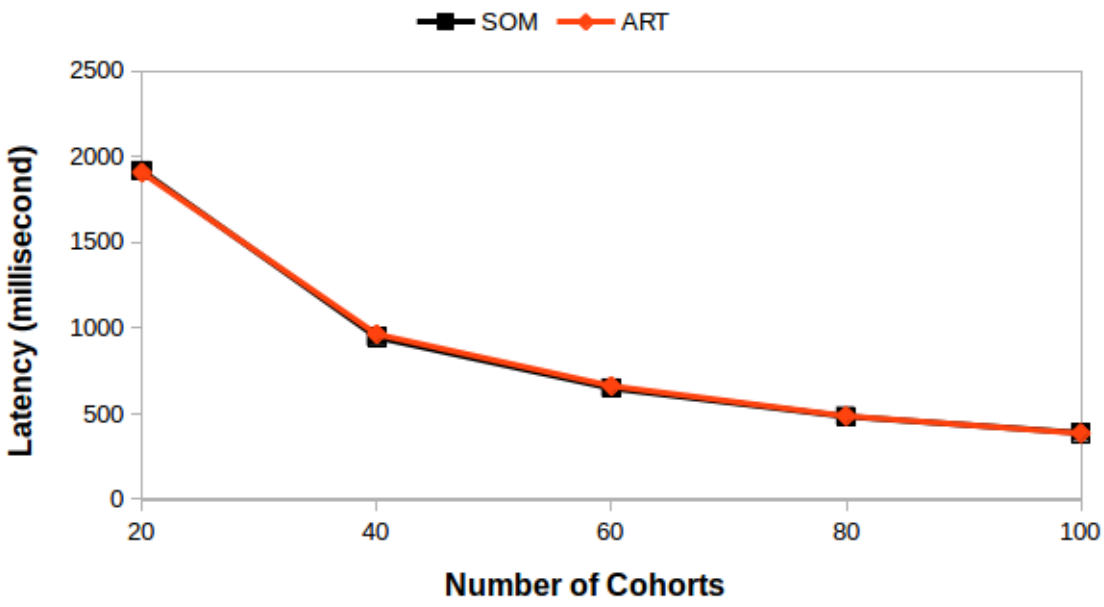


Figure 6.7: Latency in milliseconds vs. different number of cohorts based on KNN. Dataset: Physical activity monitoring features.

and Godzilla was not significant; for reference compare the RMSE of Godzilla vs Mean RMSE, and Godzilla RMSE vs Pythia RMSE. In both variants of Pythia, when the number of cohorts increased, the RMSE decreased; particularly in SOM, when 100 cohorts were used, the RMSE became very close to that of Godzilla's. The same observation was noted when EM was used; see Figure 6.11. This indicates that Pythia has comparable RMSE with

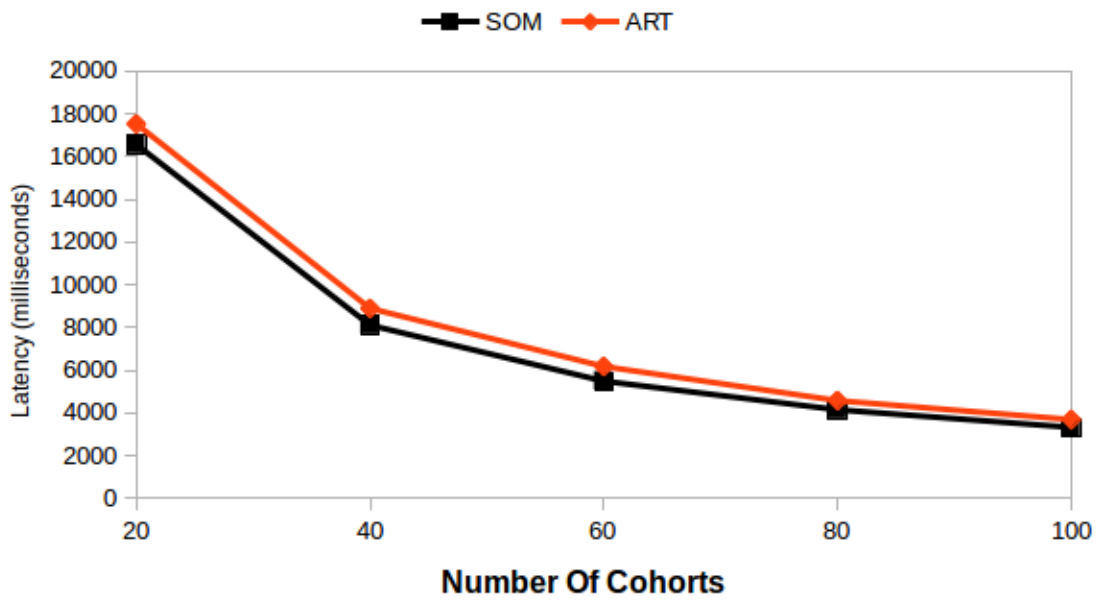


Figure 6.8: Latency in milliseconds vs. different number of cohorts based on EM. Dataset: Gas sensor array temperature modulation.

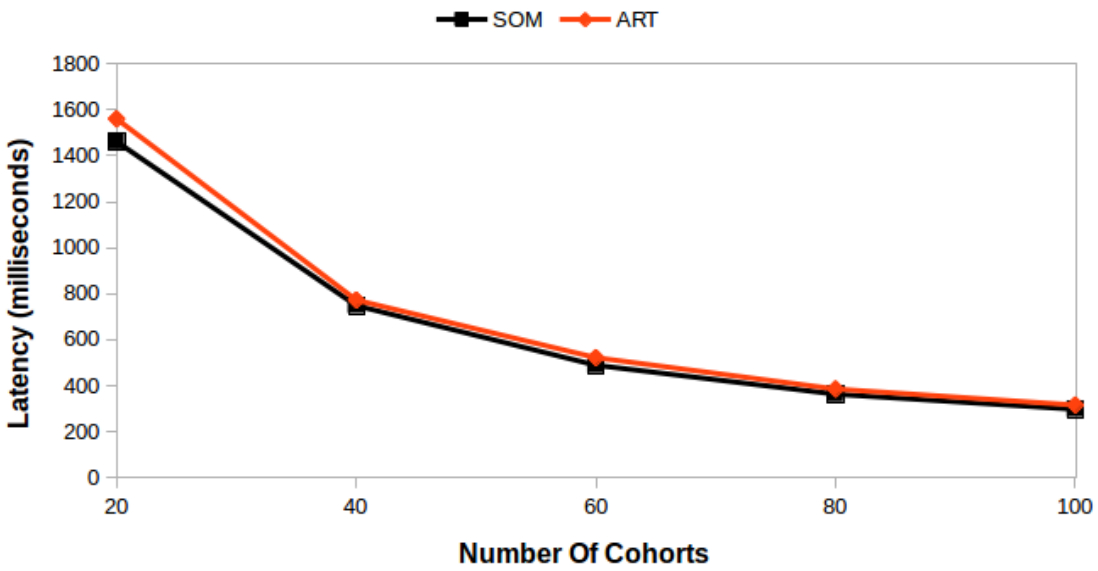


Figure 6.9: Latency in milliseconds vs. different number of cohorts based on KNN. Dataset: Gas sensor array temperature modulation.

Godzilla irrespective of the imputation algorithms and the signature creation algorithms. In contrast, for the 20-dimensional dataset, similar perfomance was noted as shown in fig. 6.12 and fig. 6.13 for kNN and EM, respectively; to avoid repetition the discussion is omitted.

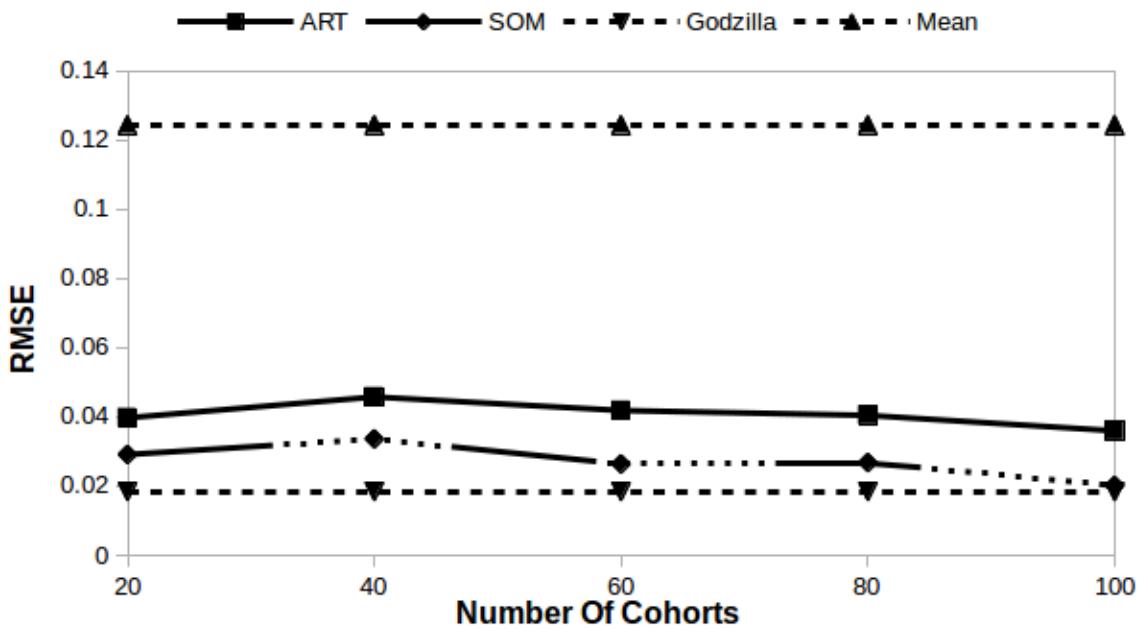


Figure 6.10: RMSE vs. number of cohorts m for Pythia ART, Pythia SOM and Godzilla using KNN. Dataset: Physical activity monitoring features .

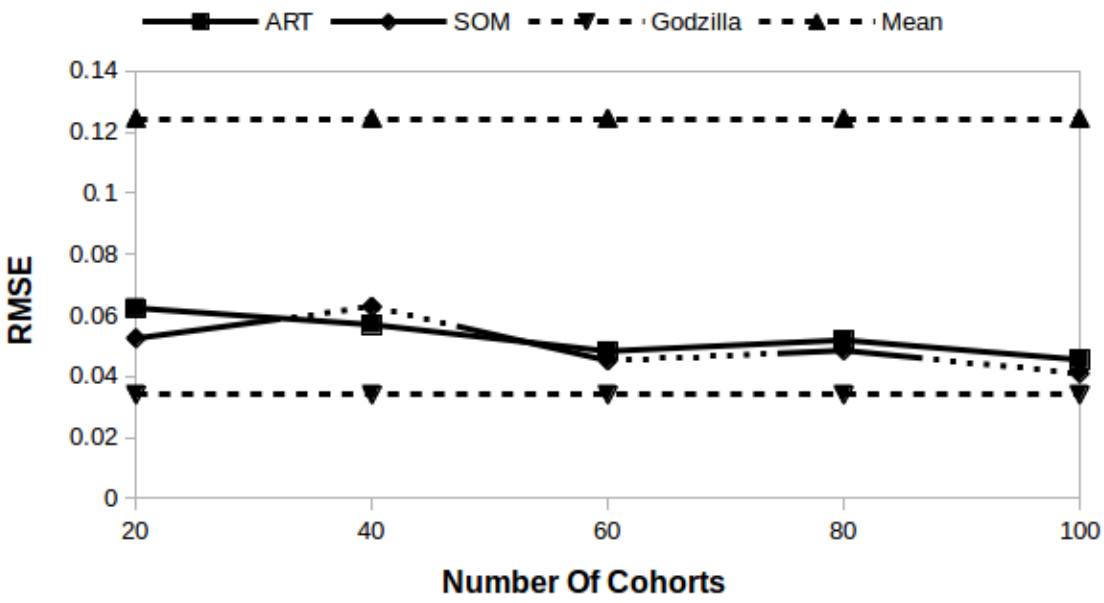


Figure 6.11: RMSE vs. number of cohorts m for Pythia ART, Pythia SOM and Godzilla using EM. Dataset: Physical activity monitoring features.

6.6.5 When accessing only relevant clusters

So far, we have seen that accessing only a relevant cohort had comparable RMSE to that of Godzilla. Furthermore, as more cohorts added to Pythia, both the latency and the estimation accuracy of Pytha improved significantly. However, considering the ever-growing data,

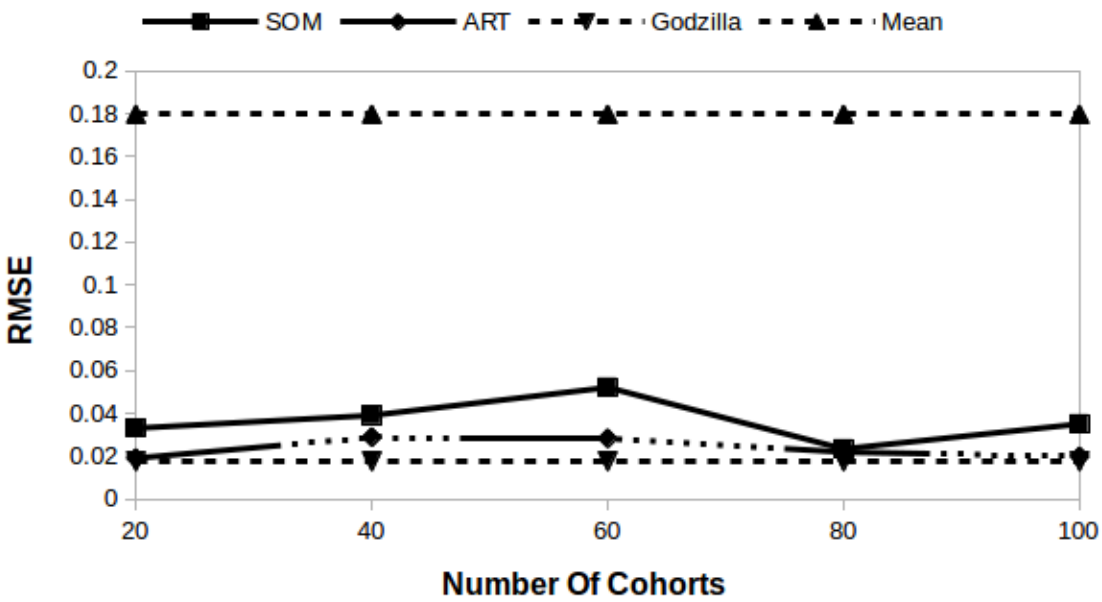


Figure 6.12: RMSE vs. number of cohorts m for Pythia ART, Pythia SOM and Godzilla using KNN. Dataset: Gas sensor array temperature modulation.

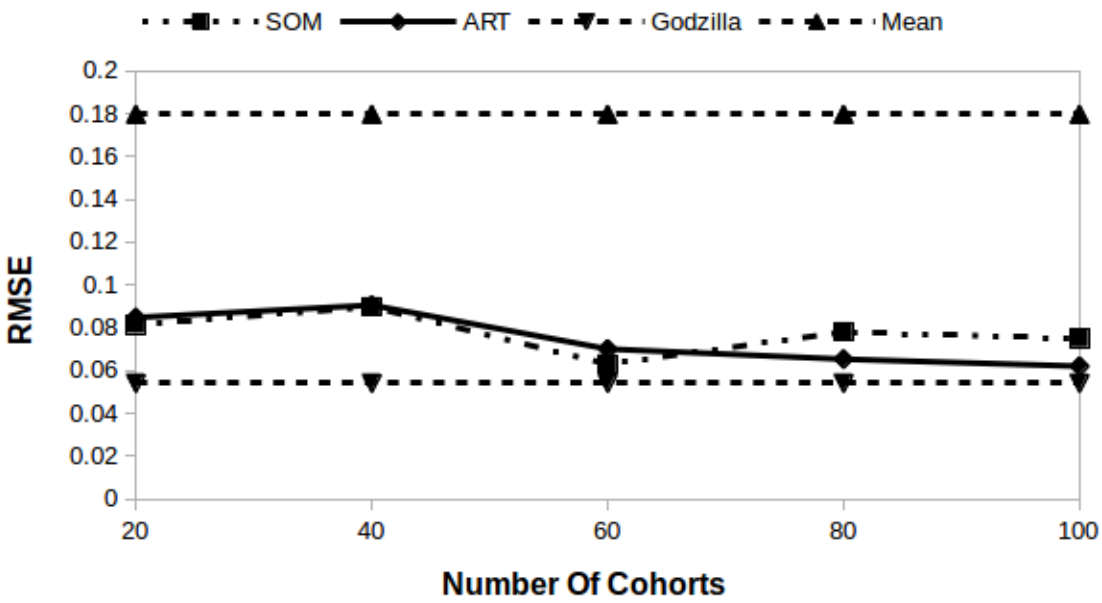


Figure 6.13: RMSE vs. number of cohorts m for Pythia ART, Pythia SOM and Godzilla using EM. Dataset: Gas sensor array temperature modulation.

adding more cohorts in order to improve the performance of Pyhta is not economically feasible. Thus, in this section, the performance of accessing only relevant clusters that reside in relevant cohorts was evaluated, for the the 51-dimensional data, as shown in and fig. 6.14 and fig. 6.15. When the estimation accuracy of accessing best cohort schema was compared to that of accessing the relevant clusters, across different number of cohorts and with ART and

SOM based signatures, no significant difference of the RMSE was noted, as shown fig. 6.14. However, the imputation latency of the best cohort schema was considerably larger than the relevant clusters schema, as shown in fig. 6.15. Furthermore, the imputation latency of the later approach was small in all the m cohorts and most importantly it did not increase with the size of the dataset, when the number of cohorts decrease. This indicates that accessing relevant clusters exceptionally scales well irrespective of number of cohorts. The same observations were noted when the 20-dimensional data was used as shown in fig. 6.16 and fig. 6.17 for estimation accuracy and latency, respectively.

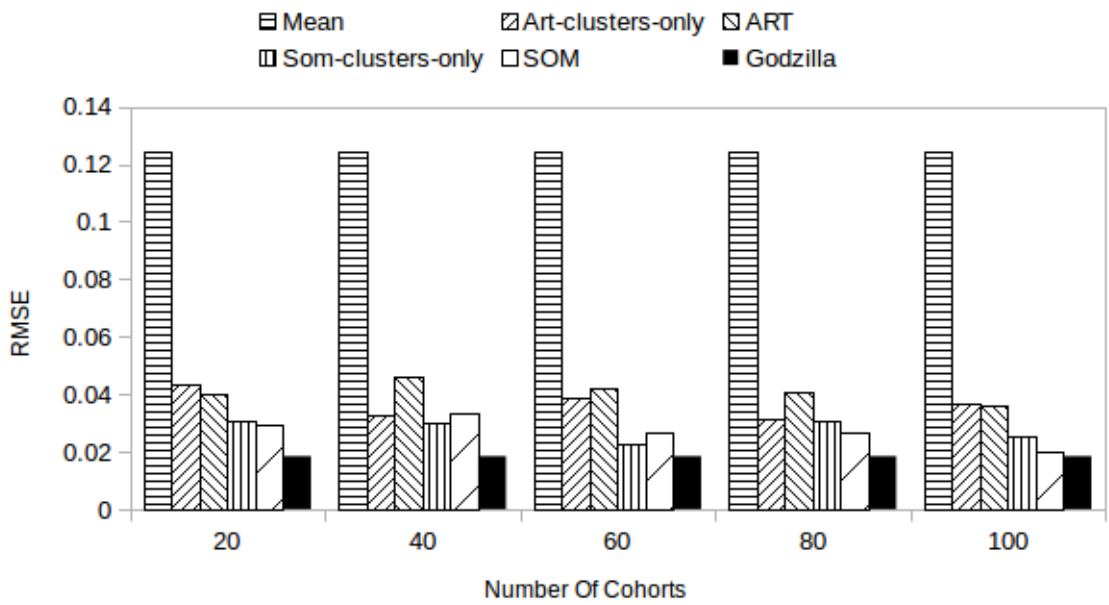


Figure 6.14: RMSE, accessing the best cohort vs the relevant clusters. Dataset: Physical activity monitoring features.

6.6.6 The effect of different distance metrics on kNN in high dimensional space

In all the previous experiments, the distance between an input vector and data elements of the datasets were computed on the basis of the Euclidean distance. However, in high dimensional space, the Euclidean distance metric might change in some non-obvious ways [3]; thus, [17] noted that the relative contrast of the distance of an input vector and a data element depends heavily on the adopted L_n distance metric where $L_n = \left(\sum_{i=1}^{d} (\mathbf{i}_i - \mathbf{c}_i)^n\right)^{1/n}$. Hence, the effect of L_n in the kNN is investigated by comparing RMSE of Pythia (or Godzilla) when the Euclidean distance and the Manhattan distance metrics were used. For the 51-dimensional dataset, as shown in fig. 6.18, almost in all cases, Manhattan distance based approached showed slightly lower RMSE, but the difference was not significantly large. However, for

Figure 6.15: Latency in milliseconds, accessing the best cohort vs. the relevant clusters. Dataset: Physical activity monitoring features .

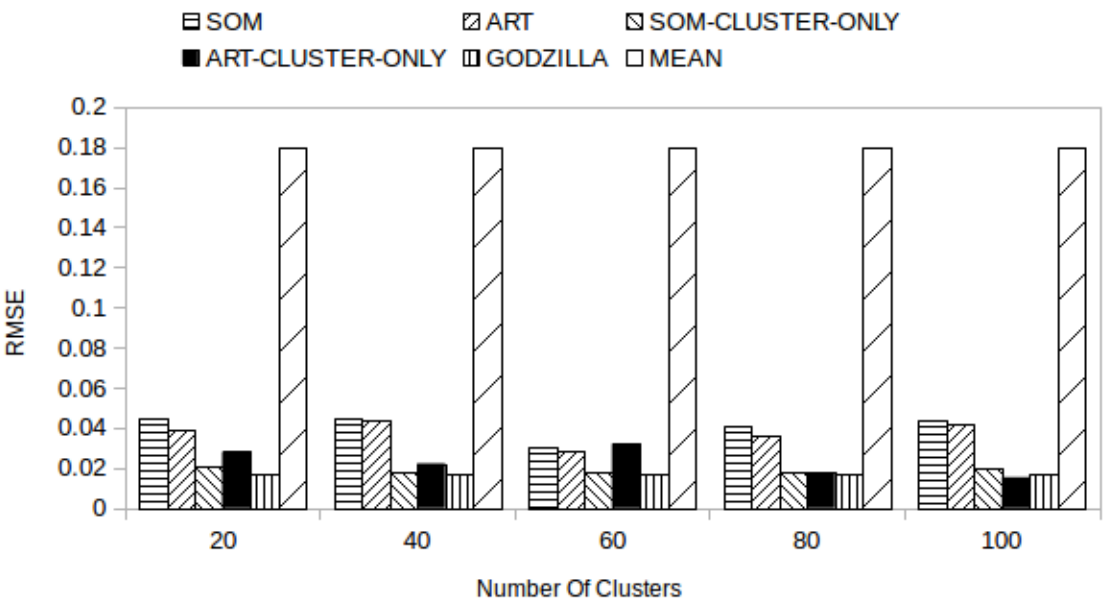


Figure 6.16: RMSE, accessing the best cohort vs the relevant clusters. Dataset: Gas sensor array temperature modulation.

the 20-dimensional dataset, see fig. 6.19, mixed results were observed. On the basis of these results, Manhattan distance might be best suited in higher dimensions; however, in future work, it is worth to investigate the effect of such distance metrics in higher dimensions than used in these experiments.

Figure 6.17: Latency in milliseconds, accessing the best cohort vs. the relevant clusters. Dataset: Gas sensor array temperature modulation.

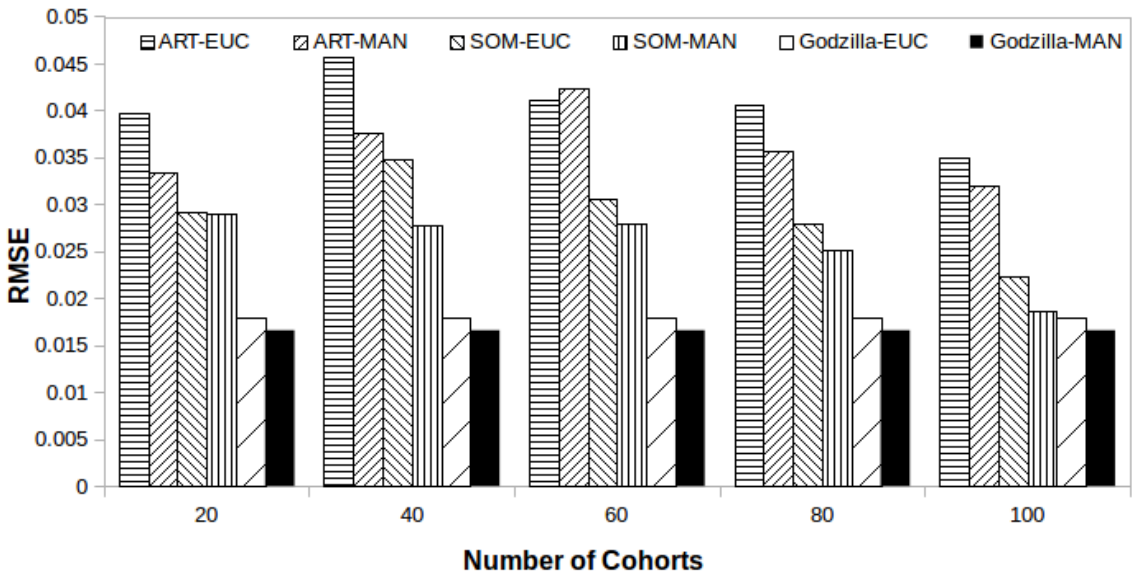


Figure 6.18: RMSE, Euclidean distance vs. Manhattan distance. Dataset: Physical activity monitoring features .

6.7 Conclusions

Pythia was originally designed to tackle the scaling out of MV problem in large-scale datasets. The Pythia has two notable features: (1) it avoids the need to access all cohorts, machines, and all associated costs for communication and for running MVAs at all cohorts; and (2)

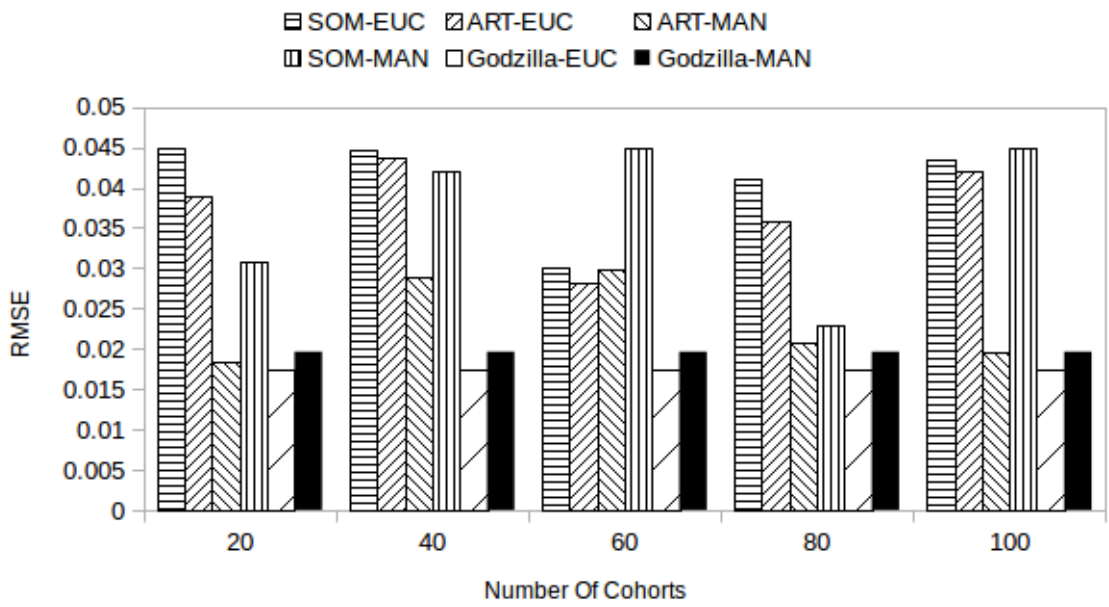


Figure 6.19: RMSE, Euclidean distance vs. Manhattan distance. Dataset: Gas sensor array temperature modulation.

achieves comparable MV imputation accuracy compared to a centralised solution. The fundamental pillars for gaining such achievement rests upon the signatures: a statistical learning structure over the distributed data. In a loose definition, signatures are like an index structure that can be exploited to identify relevant machines in order to impute MVs of an input vector. To this end, in this chapter, the performance (w.r.t estimation accuracy and MV imputation time) of Pythia is carefully studied under two different signature building algorithms: SOM and ART. The experimental results show that the accuracy of the estimated values and time elapsed when imputing MVs of Pythia, under SOM and ART based signatures, are almost identical. So, Pythia is (i) is a robust MV framework, as irrespective of the two algorithms produce different signatures, the performance of Pythia remains identical; and (ii) independent of missing values algorithms: experimental results show that the EM and kNN (two MVA) have no large difference in RMSE comparing to that of Godzilla.

As already mentioned, Pythia access few but relevant cohorts when imputing MVs. As the cost of accessing an entire dataset that resides in a data node of Pythia can be considerably expensive, even in parallel, a new method is proposed. The proposed method, instead of accessing entire data that reside in relevant cohorts, it accesses only relevant clusters; this ensures that the scalability of the kNN can be achieved without the need to add high number of cohorts to Pythia. As explained in-depth throughout this thesis, *why should an algorithm- as in case of the original design of Pythia, communicate, and process significantly large dataset, while it is possible to retrieve few data elements and still can get comparable estimation accuracy and considerably lower imputation time?*

In case of high dimensional data, the Euclidean distance metric might change in some non-obvious ways [3]; thus retrieving kNN becomes ill-defined. However, the experimental results show that the kNN MVA does not produce random results as it has a comparable estimation accuracy to that of EM and better estimation accuracy to that of mean MVA. Having said that as previous study [17] noted that the relative contrast of the distance of an input vector $\mathbf{i}$ with another vector $\mathbf{c}$ depends heavily on the adopted distance metric in high dimensional space. To this end, how Pythia would be affect by distance metrics, particularly the Euclidean distance vs Manhattan distance, when identifying relevant cohort and computing MVs using kNN is studied. Pythia, in high dimensional data space, scored slightly better accuracy when Manhattan distance metric was used but further investigation is required as the difference is small.

Chapter 7

Conclusions

Parallel and distributed approaches have proposed for large-scale data processing and have been effectively solving many data analytics problems in the big data environment. For example, Hadoop MapReduce [31] and Spark [92] are widely used in academia and industry for processing a large-scale dataset.

However, Hadoop MR/Spark is not a panacea for all large-scale data analytics. This is because of (1) there exist sophisticated data analytics approaches that are not scalable in MR/ Spark; and (2) for some data analytics problems, not all data elements stored in all the machines of a cluster are relevant. It may very well be the case that several machines contain data that has nothing to contribute to the final answer to a query. However, engaging such machines in the query processing might adversely affect query response time. For example, to retrieve only k data elements, when kNN algorithm runs in vanilla MR/Spark, an entire dataset is accessed in parallel. However, accessing a whole dataset to retrieve only k data points has not only long query response time but also wastes precious resources that otherwise can be used for other useful purposes. Consequently, processing kNN algorithm using vanilla MR/Spark is not efficient.

7.0.1 The need for accessing small part of a dataset

Instead of accessing the entire dataset, when a query is answered by accessing a small partition of a dataset that resides in a few machines, the remain machines instead of *sitting idle* can process the next query in a queue. Such a design ensures intra-query and inter-query parallelisms that are essential factors for increasing system throughput. Therefore, in this thesis, several methods that access the smallest possible data partition when executing a kNN query are proposed. To this end, this thesis tries to answer the following important question: *how to scale kNN query processing over large-scale datasets using statistical learning?*

Prior to the methods proposed in this thesis, several approaches were proposed in the literature in order to avoid accessing a whole dataset that resides in a cluster; for example, Hadoop MR based SHadoop [35] and Spark based Simba [87] are, arguably, the most popular solutions. SHadoop, for example, using multi-dimensional index approaches build global and local indexes. During query processing time, the global index is used to prone out irrelevant data partitions and a local index is used to access only relevant data elements from a partition. Even though, at query time, both solutions significantly reduce irrelevant data accesses than vanilla Hadoop MR and Spark, yet SHadoop and Simba access at least a block size of data from DFS (distributed file system). This is due to, in both methods, a lower size of a data partition is determined by a lower size of a block in which they operate; for example, when they run over Hadoop distributed file system (HDFS) this translates to 128MB (a minimum block size of HDFS). This implies that during query execution to retrieve k data elements - keep in mind that most of the time the value of k is small- few tens of millions of data points,if not hundredth of millions, have to be loaded from DFS to the memory. Due to this high number of irrelevant data access, such a design has high query response time.

7.0.2 Coordinator with and without in-Memory Index Tree

To tackle such as problem, in this thesis, a coordinator-based approach (called COWI) is proposed. COWI adopts Quad-tree (QT) to partitions a large-scale dataset into very small partitions(aka cells). Furthermore, in COWI there is no limit on the lower size of a partition as data is stored in the NoSQL key-value data store HBase [41]. Thus during query processing time, by accessing only a few data elements, COWI has up to *three order magnitude lower query response time* than SHadoop and Simba.

The critical point put forward here is that system reliability should not coined with query response time. SHadoop or Simba may ensure system reliability (health of a cluster by not overloading the NameNode with too many meta-data of small partitions) but unnecessarily sacrifice query efficiency. Experimental results of COWI demonstrate that computing a kNN query in low-dimensional data should and could be a matter of a few tens of milliseconds and not several (tens of) seconds which is the case with state-of-the-art Shadoop and Simba. COWI by paying attention to (i) data organisation, (ii) storage, and (iii) indexing in a way that allows access to only relevant data points; thus it manages to process kNN query in few milliseconds; so: *why should an algorithm- as in case of Shadoop and Simba- retrieve from storage, communicate, and process millions of data items, while it is possible to retrieve a few thousand (if not hundredth) of data elements when processing a* 10*NN query?*

In COWI, when a large-scale dataset is indexed using QT, the resulting tree-like data structure is assumed to be stored in memory to serve as an index. However, when a dataset is partitioned into tiny cells, a size of the resulting index (tree-like data structure) -that is

supposed to be stored in memory- increases significantly. Particularly, when indexing a very large dataset, *the size of the index might exceed the available free memory in the coordinator*. In such a case, a size of a cell- the smallest data partition in COWI- can be increased to reduce a size of an index to fit in the available memory in the coordinator. However, *raising a size of a cell increases the chance of accessing irrelevant data elements* to a query and hence has an adverse effect on query response time. To tackle this problem CONI is proposed.

CONI stores part of an index in a key-value data store in HBase table. During query execution, in CONI, two HBase tables are accessed: (i) a table that contains an index, and (ii) a table that maintains a dataset. Consequently, in CONI, irrespective of the amount of free available memory in the coordinator, a dataset can be partitioned into very small cells; as such accessing irrelevant data points reduces significantly. But as accessing index from HBase table that supposed to be stored in a memory is relatively expensive, the experimental results showed that *CONI has relatively higher query response time than COWI, but yet has lower query response time than Simba and SHadoop*.

In short, when multi-dimensional indexing methods are used to index a dataset, the tree-like data structure has to be stored either in memory or disk to serve as index. In systems that store the index in memory, such as COWI, since infinity memory cannot be assumed at the coordinator and nowadays a size of a data is growing at an alarming rate, a lower size of a cell, sooner or later, will be inevitably limited by the size of the memory in the coordinator. On the other hand, storing an index in HBase table, as in the case of CONI, has relatively higher query response time.

7.0.3 STOS

For ensuring optimal query response time irrespective of a size of a dataset and available memory in the coordinator, a novel method (named STOS) is proposed in this thesis. Drafting away from the tradition tree-based multi-dimensional indexing approaches, STOS partitions a large-scale dataset using statistical learning methods namely: Gaussian Mixture Models [24], Probability integral transformation [22] and Independence copula [40]. STOS transforms an arbitrary distribution of a dataset into a standard multivariate (mv) uniform distribution. Hereinafter, STOS builds a simple uniform grid for partitioning a large-scale dataset based on the uniformly distributed data. Experimental results show that *STOS has the same query response time to that of COWI, several orders of magnitude lower memory footprint, a memory footprint that is practically independent of the dataset size, and easy and fast recoverability*. The minute size of STOS allows for several copies of it to be stored in different places, thus allowing the system to be up and running quickly after failures.

In STOS, a lower size of a cell is determined based neither on an available free memory in the coordinator nor does on a block size of a DFS. STOS only takes into consideration what

is best for query response time when determining size of cells; thus, STOS can partition a dataset into a very small cells. Consequently, *STOS is an ideal solution for reducing irrelevant data access during kNN query processing*.

In contrast, as multi-dimensional tree-based indexing approaches were originally designed to run in a centralised system, they do not work in parallel/distributed systems easily. To this end, several methods were proposed for adopting such indexing methods in parallel/distributed systems. However, all the proposed approaches, according to a recent survey [73], suffer from several significant drawbacks: (1) fail to address efficiency and scalability problems that result from increasing index sizes; (2) struggle to index non-uniform big data; or (3) face problems related to Either poor data proximity preservation or poor applicability in higher dimensions (NB in this context higher dimensions means usually less than 10 to 15 dimensions).

In this thesis, therefore, *the performance of STOS, as data partitioning method over large-scale datasets, is compared to tree-based approaches, particularly to QT*. The experimental results indicate that STOS

- has several orders of magnitude lower indexing building time,
- has several orders of magnitude lower memory footprint,
- has better data proximity preservation, and
- has better applicability in higher dimensional data.

Unfortunately, it should be noted that the experimental results also show that similar to tree-based index approaches, query response time of STOS increases with dimensions. This implies that the number of irrelevant data points that are accessed during query processing time increases with the number of dimensions of a dataset. In the literature, this problem is a.k.a the curse of dimensionality. In this thesis, to tackle with such problem, a novel kNN processing (called estimatedd kNN query processing) is proposed.

7.0.4 Estimated kNN and Probabilistic kNN regression

The estimated kNN processing barrows its design philosophy from approximated kNN query processing. The key design philosophy of approximated kNN query processing states that *approximate nearest neighbour is almost as good as the exact one, especially when the distance measure accurately captures the notion of user quality, then small differences in the distance should not matter* [9]. Based on this principle, unlike to approximated kNN, estimated kNN statistically predicts locations of kNN of a given query without accessing the

dataset. For accurately capturing the notion of users quality, the proposed approach receives a user-defined distance - an average distance between actual kNN and estimated kNN a user is willing to tolerate; afterwards *estimated kNN processing approach can notify the user a degree of confidence that the estimated kNN can be located within the given distance*.

The estimated kNN processing method uses space transformation techniques for predicting kNN. The statistical parameters that are used by STOS for transforming an arbitrary distribution of a dataset into multivariate uniform distribution can be directly reused by the estimated kNN processing approach. Furthermore, a degree of confidence in which the estimated kNN are located within a users defined-distance is computed based on $\mathcal{X}^2$ distribution.

Experimental results reveal that the estimated kNN has (i) a high degree of accuracy, (ii) achieves orders of magnitude lower query response time especially in higher dimensional data, (iii) query response time that is practically independent to a number of dimensions of a dataset and (iv) no disk and network I/Os when estimating kNN.

The other attractive feature of the estimated kNN processing method is its flexibility to work as a stand-alone solution or to work in conjunction with STOS (when the number of dimensions is reasonably small). As all statistical parameters that are used for space transformation in STOS can directly be re-used in the estimated kNN processing method, both solution can work side by side efficiently without any extra overhead cost. Consequently, a user can enjoy the flexibility of choosing either to wait for a longer time and get exact kNN answer or to get quickly estimated results with high accuracy. Moreover, as the estimated kNN has a minute memory footprint, it can be deployed to a client-side. This has a significant impact on (i) reducing a workload of the coordinator and (ii) reducing network traffic. To the best of the author's knowledge, this is the first step for dataless big data analytics.

In the same venue, as kNN regression is based on kNN, in this thesis, a new kNN regression method is proposed. The proposed method is affectionately named probabilistic kNN regression. A vanilla kNN regression identifies kNN based on the distance between predictor random variables (RVs) of the query and the dataset. Then, vanilla kNN regression predicts a value of the target random variable (RV) of the query by averaging values of the target RV of the kNN that are retrieved from a dataset. However, without accessing the underlying dataset, probabilistic kNN predicts k plausible values of the target RV and then it computes average value of the k predicted values in order to estimate the target RV of the query. The proposed method, like the estimated kNN query processing, re-used all the statistical parameters that are used by STOS for transforming data into multivariate uniform distribution. Experimental results showed that the probabilistic kNN query has a high prediction accuracy and obviously has lower query response time. *Even though, further research is needed, probabilistic kNN has shown a promising results towards breaking the curse of dimensionality when computing kNN regression*.

7.0.5 kNN as missing value imputation algorithm in high dimensional space

Last but not least, kNN as a missing value algorithm (MVA) in high dimensional space in Pythia [8]- a framework of missing value imputation- is studied thoroughly. Bear in mind that Pythia does not guarantee to retrieve exact kNN. Thus, different space quantisation (loosely can be defined as indexing high dimensional data, has no limit on number of dimensions as in the case of tree-based indexing approaches) methods are compared to investigate whether the accuracy of kNN when data is partitioned by the self-organising maps (SOM) is better than when data is partitioned by adaptive resonance theory (ART). Experimental results shows that scalability and performance of kNN under the two space quantisation techniques have no significant difference. Furthermore, the accuracy of kNN as MVA in high dimensional space is studied by comparing to a more sophisticated MVA the expectation-maximisation (EM). In spite of its simplicity to implement, kNN has comparable or better performance compared to EM, but, intuitively, kNN has significantly lower processing imputation time. Pythia access few but relevant data nodes when imputing MVs. However,as the cost of accessing an entire dataset that resides in a data node is considerably expensive, even in parallel, a new method is proposed in this thesis. The proposed method, instead of accessing entire data that reside in relevant data nodes, it accesses only relevant clusters. Therefore, *why should an algorithm- as in case of the original design of Pythia, communicate, and process significantly large dataset, while it is possible to retrieve few data elements and still can get comparable results?* Last but not least, the effect of distance metrics in the high dimensional space of kNN as MVA in Pythia is studied in this thesis by comparing the accuracy of kNN algorithms based on Euclidean vs Mahnabolis distance metrics. In both cases, kNN has almost the same accuracy but it is worth to note that Mahnabolis based kNN have shown slightly better accuracy.

7.1 Lessons Learned

Major lessons learned in this thesis are summarised as follows :

- for scaling out kNN queries over large-scale datasets, a system has to support inter-query and intra-query parallelism;
- during kNN query processing, by avoiding *irrelevant* data access not only reduces query response time but also contributes for efficient utilisation of resources, which in turn maximises inter-query and intra-query parallelism that has a tremendous effect on increasing system throughput;

- when partitioning a large-scale dataset for kNN processing -as a block size of a DFS is usually large- the determination of a lower size of a partition by a block size of a DFS increases irrelevant data access and thus should be avoided;
- when partitioning large-scale datasets, existing multi-dimensional indexing approaches suffer from efficiency or scalability issues and this opens the space to incorporate more complicated techniques;
- even though storing part of the index in DFS can alleviate the efficiency or scalability problem of existing multi-dimensional indexing approaches, it adversely affects the query response time;
- an efficient and scalable large-scale data partitioning method can be built using statistical space transformation techniques;
- an index built based on space transformation technique, has the comparable query performance as the existing multi-dimensional indexing approaches, better index building time, comparable or better space utilisation, extremely low memory footprint, easy and fast recoverability, and exceptionally scalable as a memory footprint of the index is independent of a dataset size;
- space transformation technique can be exploited to estimate kNN with high speed and accuracy;
- the same space transformation technique can be used for predicting a value of a target RV of a kNN regression without computing distance and hence has a promising application towards breaking the curse of dimensionality;
- as the statistical parameters used for space transformation have a minute memory footprint, estimating kNN and probabilistic kNN regression methods can be deployed to a client-side; this does not only reduce a workload on the server and reduce network traffic but also opens a way towards dataless big data analytics;

7.2 Future Work

This thesis mainly demonstrates that scaling kNN queries in large-scale datasets depends on reducing irrelevant data access during query execution. By revisiting the design philosophy that underpins kNN query processing over very large datasets, going against the grain and state of the art, and exploiting statistical learning methods, a novel way of indexing and organising a dataset that enables accesses to only very small subsets of the dataset is proposed. The proposed methods can lead to several future works that are listed below.

Other Types of Queries: The superiority of the proposed method for computing kNN over large-scale datasets is demonstrated using extensive series of experiments. In future, investigating the performance of the proposed indexing method for other queries, such as join queries, is essential. Furthermore, like the estimated kNN and probabilistic kNN, it might be interesting to come up with estimated join query or probabilistic join query methods based on statistical space transformation methods.

Furthermore, as the proposed space transformation method can represent data in three different domain (data) spaces -original, independent, uniform domain spaces- possibly, this can lead the way for a private kNN query processing. It is common to store data in a public cluster and hence storing a data in a different representation, e.g in the form of a uniform distribution, can mask a data from direct exposure. Conquently, the proposed method can be studied further in the perspective of private kNN query processing.

Another important query that can be solved using space transformation technique is a reverse kNN. In reverse kNN query, all data points that have a given query in their kNN are retrieved. As most of the existing methods retrieve only approximate results, STOS should be studied on the perspective of exact reverse kNN processing; the Uniform domain space might be exploited for retrieving exact reverse kNN. Furthermore, in reverse kNN, instead of retrieving approximate results, it is worthwhile to investigate the idea of estimating reverse kNN in the same way as the estimated kNN processing. Last but not least, related queries such as Reverse Farthest Neighbor might be processed efficiently using space transformation techniques.

Data Stream Environments: In this thesis, data is assumed to be stationary -the probability distribution function does not change over the time. Hence, all the statistical parameters learned from the dataset either in Pythia, STOS, estimated kNN or probabilistic kNN remain stable. In non-stationary data, however, a distribution of dataset changes over time swiftly and thus new clusters can be created, while, existing clusters might perish. Therefore, in the future, it is worthwhile to try Pythia, STOS, estimated kNN and probabilistic kNN to adapt in non-stationary data environment.

Missing Value Imputation: In kNN based missing value imputation, kNN data points are identified based on the distance between the observed values of the input data and elements of a dataset. It might be a good idea to try probabilistic kNN regression for missing value imputation since it predicts k plausible values of a target RV without accessing the dataset.

Abbreviations

ART	Adaptive Resonance Theory
Abalon	The Population Biology of Abalone Haliotis species in Tasmania
Air	Air Quality
AkNN	Approximated kNN
BIC	Bayesian information criterion
CCPP	The Combined Cycle Power Plant
CONI	Coordinator With No Index
COWI	Coordinator With Index
CV or cv	Coefficient of variation
DFS	Distributed File System
EM or em	Expectation Maximization
GMM	Gaussian Mixture Models
HDFS	Hadoop Distributed File System
MAE	Mean absolute error
MAR	Missing at Random
MBR	Minimum bounding (hyper)Rectangle
MCAR	Missing completely at random
MLE	Maximum Likelihood estimation
MNAR	Missing Not at Random
MR	Map Reduce
MV	Missing value
MVA	Missing value algorithm
NN	Nearest neighbour
PIT	Probability integral transformation
QT or Q-T	Quad Tree
RDD	Resilient distributed datasets
RMSE	Root mean square error
RV	Random Variable
SH	Spatial Hadoop
SH-HDFS	Spatial Hadoop direct acces to Hadoop Distributed File System

SH-MR	Spatial Hadoop with MapRduce
SHadoop	Spatial Hadoop
SOM	Self-Organizing Maps
STOS	Space Transformation Organisation Structure
STR	Sort-TileRecursive
cdf	Cumulative distribution function
ecdf	Empirical cumulative distribution function
ind	Independent
kNN	k-nearest neighbour
m-d	Multi-dimensional
ms	Milliseconds
mv	Multivariate
org	Original
pdf	Distribution density function
uni	Uniform

Bibliography

[1] The data challenge. `http://www.eiroforum.org/activities/seientific_highlights/201209_XFEL/index.html`.

[2] Twitter, the about webpage. `https://about.twitter.com/company`.

[3] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[4] T. Aittokallio. Dealing with missing values in large-scale studies: microarray data imputation and beyond. *Briefings in bioinformatics*, 11(2):253–264, 2009.

[5] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop GIS: A high performance spatial data warehousing system over MapReduce. *PVLDB*, 6(11):1009–1020, 2013.

[6] O. Akbilgic, H. Bozdogan, and M. E. Balaban. A novel hybrid rbf neural networks model as a forecaster. *Statistics and Computing*, 24(3):365–375, 2014.

[7] A. M. Aly, A. S. Abdelhamid, A. R. Mahmood, W. G. Aref, M. S. Hassan, H. Elmeleegy, and M. Ouzzani. A demonstration of AQWA: Adaptive query-workload-aware partitioning of big spatial data. *PVLDB*, 8(12):1968–1971, 2015.

[8] C. Anagnostopoulos and P. Triantafillou. Scaling out big data missing value imputations: pythia vs. godzilla. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 651–660. ACM, 2014.

[9] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[10] M. Armbrust et al. Spark sql: Relational data processing in spark. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 1383–1394, 2015.

[11] M. Atallah. Algorithms and theory of computation handbook 1999. *Google Scholar Google Scholar Digital Library Digital Library*.

[12] I. B. Aydilek and A. Arslan. A novel hybrid approach to estimating missing values in databases using k-nearest neighbors and neural networks. *International Journal of Innovative Computing, Information and Control*, 7(8):4705–4717, 2012.

[13] X. Bao, L. Liu, N. Xiao, F. Liu, Q. Zhang, and T. Zhu. Hconfig: Resource adaptive fast bulk loading in hbase. In *Proc. IEEE Intl. Conf. on Collaborative Computing (CollaborateCom)*, pages 215–224, 2014.

[14] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, 1990.

[15] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[16] J. L. Bentley, D. F. Stanat, and E. H. Williams. The complexity of finding fixed-radius near neighbors. *Information processing letters*, 6(6):209–212, 1977.

[17] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[18] A. Cahsai, N. Ntarmos, C. Anagnostopoulos, and P. Triantafillou. Scaling k-nearest neighbours queries (the right way). In *Proc. IEEE Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 1419–1430, 2017.

[19] S. Cambanis, S. Huang, and G. Simons. On the theory of elliptically contoured distributions. *Journal of Multivariate Analysis*, 11(3):368 – 385, 1981.

[20] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, Mar. 1988.

[21] A. Cary, Y. Yesha, M. Adjouadi, and N. Rishe. Leveraging cloud computing in geo-database management. In *Granular Computing (GrC), 2010 IEEE International Conference on*, pages 73–78. IEEE, 2010.

[22] G. Casella and R. Berger. Statistical inference, brooks/cole pub. *Co, Pacific Grove, CA*, 1990.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[24] J. Chen. Optimal rate of convergence for finite mixture models. *The Annals of Statistics*, pages 221–233, 1995.

[25] E. C. Chi, H. Zhou, G. K. Chen, D. O. Del Vecchyo, and K. L. Lange. Genotype imputation via matrix completion. *Genome research*, pages gr–145821, 2012.

[26] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

[27] W. J. Cody. Rational chebyshev approximations for the error function. *Mathematics of Computation*, 23(107):631–637, 1969.

[28] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[29] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008.

[30] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[31] J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[33] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[34] A. Eldawy, L. Alarabi, and M. F. Mokbel. Spatial partitioning techniques in SpatialHadoop. *PVLDB*, 8(12):1602–1605, 2015.

[35] A. Eldawy and M. F. Mokbel. A demonstration of SpatialHadoop: An efficient MapReduce framework for spatial data. *PVLDB*, 6(12):1230–1233, 2013.

[36] L. T. Ene, E. Næsset, and T. Gobakken. Model-based inference for k-nearest neighbours predictions using a canonical vine copula. *Scandinavian journal of forest research*, 28(3):266–281, 2013.

[37] A. Farhangfar, L. Kurgan, and J. Dy. Impact of imputation of missing values on classification error for discrete data. *Pattern Recognition*, 41(12):3692–3705, 2008.

[38] A. Farhangfar, L. A. Kurgan, and W. Pedrycz. A novel framework for imputation of missing values in databases. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(5):692–709, 2007.

[39] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[40] C. Genest and A.-C. Favre. Everything you always wanted to know about copula modeling but were afraid to ask. *Journal of hydrologic engineering*, 12(4):347–368, 2007.

[41] L. George. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.

[42] S. Ghemawat, H. Gobioff, and S.-T. Leung. *The Google file system*, volume 37. ACM, 2003.

[43] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, pages 47–57, 1984.

[44] D. Han and E. Stroulia. Hgrid: A data model for large geospatial data sets in hbase. In *Proc. IEEE Intl. Conf. on Cloud Computing (CLOUD)*, pages 910–917, 2013.

[45] Y.-T. Hsu, Y.-C. Pan, L.-Y. Wei, W.-C. Peng, and W.-C. Lee. Key formulation schemes for spatial index in cloud data managements. In *Proc. IEEE Intl. Conf. on Mobile Data Management (MDM)*, pages 21–26, 2012.

[46] T. Huang, H. Peng, and K. Zhang. Model selection for gaussian mixture models. *arXiv preprint arXiv:1301.3558*, 2013.

[47] P. Indyk. Nearest neighbors in high-dimensional spaces. 2004.

[48] L. Jiang, B. Li, and M. Song. The optimization of HDFS based on small files. In *Proc. IEEE Intl. Conf. on Broadband Network and Multimedia Technology (IC-BNMT)*, pages 912–915, 2010.

[49] D. W. Joenssen and U. Bankhofer. Hot deck methods for imputing missing data. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 63–75. Springer, 2012.

[50] D.-W. Kim and B.-Y. Kang. Iterative clustering analysis for grouping missing data in gene expression profiles. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 129–138. Springer, 2006.

[51] T. Kohonen, M. Schroeder, T. Huang, and S.-O. Maps. Springer-verlag new york. *Inc., Secaucus, NJ*, 43(2), 2001.

[52] L. Kurgan, K. J. Cios, M. Sontag, and F. J. Accurso. Mining the cystic fibrosis data. *Next generation of data-mining applications*, pages 415–444, 2005.

[53] K. Lakshminarayan, S. A. Harp, and T. Samad. Imputation of missing data in industrial databases. *Applied intelligence*, 11(3):259–275, 1999.

[54] S. T. Leutenegger, M. A. Lopez, and J. Edgington. STR: A simple and efficient algorithm for R-tree packing. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 497–506, 1997.

[55] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 507–516. ACM, 2009.

[56] A. W.-C. Liew, N.-F. Law, and H. Yan. Missing value imputation for gene expression data: computational techniques to recover missing data from available information. *Briefings in bioinformatics*, 12(5):498–513, 2010.

[57] R. J. Little and D. B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 1983.

[58] G. J. McLachlan and K. E. Basford. *Mixture models: Inference and applications to clustering*, volume 84. Marcel Dekker, 1988.

[59] X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7, 2016.

[60] M. Minsky and S. Papert. Perceptrons: Anlntroduction to computational geometry. *MITPress, Cambridge, Massachusetts*, 1969.

[61] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. IBM, New York, 1966.

[62] L. D. NASA. Modis land products quality assurance tutorial: Part-1. *USGS EROS Center, Sioux Falls*, 2016.

[63] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. MD-HBase: Design and implementation of an elastic data infrastructure for cloud-scale location services. *Distrib. Parallel Databases*, 31(2):289–319, 2013.

[64] M. Ouyang, W. J. Welsh, and P. Georgopoulos. Gaussian mixture clustering and imputation of microarray data. *Bioinformatics*, 20(6):917–923, 2004.

[65] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli. Human activity recognition using multisensor data fusion based on reservoir computing. *Journal of Ambient Intelligence and Smart Environments*, 8(2):87–107, 2016.

[66] Y. Park, M. Cafarella, and B. Mozafari. Neighbor-sensitive hashing. *Proceedings of the VLDB Endowment*, 9(3):144–155, 2015.

[67] S. T. Rachev, C. Menn, and F. J. Fabozzi. *Fat-tailed and skewed asset return distributions: implications for risk management, portfolio selection, and option pricing*, volume 139. John Wiley & Sons, 2005.

[68] T. E. Raghunathan, J. M. Lepkowski, J. Van Hoewyk, and P. Solenberger. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey methodology*, 27(1):85–96, 2001.

[69] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. *ACM SIGMOD Record*, 24(2):71–79, 1995.

[70] D. B. Rubin. Multiple imputation after 18+ years. *Journal of the American statistical Association*, 91(434):473–489, 1996.

[71] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[72] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[73] H. Singh and S. Bawa. A survey of traditional and MapReduce-based spatial query processing approaches. *ACM SIGMOD Record*, 46(2):18–29, 2017.

[74] M. Sklar. *Fonctions de répartition à n dimensions et leurs marges*. Université Paris 8, 1959.

[75] X. Su, R. Greiner, T. M. Khoshgoftaar, and A. Napolitano. Using classifier-based nominal imputation to improve machine learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 124–135. Springer, 2011.

[76] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref. Locationspark: A distributed in-memory data management system for big spatial data. *PVLDB*, 9(13):1565–1568, 2016.

[77] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[78] P. Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

[79] D. Vohra. Apache hbase primer, 2016.

[80] H. Wang, F. Zhuang, X. Ao, Q. He, and Z. Shi. Scalable bootstrap clustering for massive data. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*, pages 1–6. IEEE, 2014.

[81] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big dataa survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[82] J. Wang, T. Zhang, N. Sebe, H. T. Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2018.

[83] S. Waugh. *Extending and benchmarking cascade-correlation. Dept of Computer Science, University of Tasmania*. PhD thesis, Ph. D. Dissertation, 1995.

[84] I. H. Witten and E. Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

[85] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[86] F. Xiao. A Spark based computing framework for spatial data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 125–130, 2017.

[87] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial analytics. In *Proc. ACM Intl. Conf. on Management of Data (SIGMOD)*, pages 1071–1085, 2016.

[88] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[89] S. Yang, K. Kalpakis, C. F. Mackenzie, L. G. Stansbury, D. M. Stein, T. M. Scalea, and P. F.-M. Hu. Online recovery of missing values in vital signs data streams using low-rank matrix completion. In *ICMLA (1)*, pages 281–287, 2012.

[90] S. You, J. Zhang, and L. Gruenwald. Large-scale spatial join query processing in cloud. In *Proc. IEEE Intl. Conf. on Data Engineering Workshops (ICDEW)*, pages 34–41, 2015.

[91] J. Yu, J. Wu, and M. Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proc. SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, page 70, 2015.

[92] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proc. USENIX Conf. on Hot Topics in Cloud Computing (HotCloud)*, pages 10–10, 2010.

[93] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[94] S. Zhang, L. Miao, D. Zhang, and Y. Wang. A strategy to deal with mass small files in HDFS. In *Proc. Intl. Conf. on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, pages 331–334, 2014.

[95] Y. Zhang and D. Liu. Improving the efficiency of storing for small files in HDFS. In *Proc. IEEE Intl. Conf. on Computer Science and Service System (CSSS)*, pages 2239–2242, 2012.