Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

IMPROVING k-NN SEARCH AND SUBSPACE CLUSTERING BASED ON LOCAL INTRINSIC DIMENSIONALITY

by
Arwa M. Wali

In several novel applications such as multimedia and recommender systems, data is often represented as object feature vectors in high-dimensional spaces. The high-dimensional data is always a challenge for state-of-the-art algorithms, because of the so-called "curse of dimensionality". As the dimensionality increases, the discriminative ability of similarity measures diminishes to the point where many data analysis algorithms, such as similarity search and clustering, that depend on them lose their effectiveness. One way to handle this challenge is by selecting the most important features, which is essential for providing compact object representations as well as improving the overall search and clustering performance. Having compact feature vectors can further reduce the storage space and the computational complexity of search and learning tasks.

Support-Weighted Intrinsic Dimensionality (support-weighted ID) is a new promising feature selection criterion that estimates the contribution of each feature to the overall intrinsic dimensionality. Support-weighted ID identifies relevant features locally for each object, and penalizes those features that have locally lower discriminative power as well as higher density. In fact, support-weighted ID measures the ability of each feature to locally discriminate between objects in the dataset.

Based on support-weighted ID, this dissertation introduces three main research contributions: First, this dissertation proposes NNWID-Descent, a similarity graph construction method that utilizes the support-weighted ID criterion to identify and retain relevant features locally for each object and enhance the overall graph quality. Second, with the aim to improve the accuracy and performance of cluster analysis,

this dissertation introduces k-LIDoids, a subspace clustering algorithm that extends the utility of support-weighted ID within a clustering framework in order to gradually select the subset of informative and important features per cluster. k-LIDoids is able to construct clusters together with finding a low dimensional subspace for each cluster. Finally, using the compact object and cluster representations from NNWID-Descent and k-LIDoids, this dissertation defines LID-Fingerprint, a new binary fingerprinting and multi-level indexing framework for the high-dimensional data. LID-Fingerprint can be used for hiding the information as a way of preventing passive adversaries as well as providing an efficient and secure similarity search and retrieval for the data stored on the cloud. When compared to other state-of-the-art algorithms, the good practical performance provides an evidence for the effectiveness of the proposed algorithms for the data in high-dimensional spaces.

IMPROVING k-NN SEARCH AND SUBSPACE CLUSTERING BASED ON LOCAL INTRINSIC DIMENSIONALITY

by
Arwa M. Wali

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology – Newark
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2018

Copyright © 2018 by Arwa M. Wali
ALL RIGHTS RESERVED

APPROVAL PAGE

IMPROVING k-NN SEARCH AND SUBSPACE CLUSTERING BASED ON LOCAL INTRINSIC DIMENSIONALITY

Arwa M. Wali

Dr. Vincent Oria, Dissertation Co-Advisor — Date
Professor, Department of Computer Science, NJIT

Dr. Michael E. Houle, Dissertation Co-Advisor — Date
Visiting Professor, National Institute of Informatics, Japan

Dr. Ali Mili, Committee Member — Date
Professor, Department of Computer Science, NJIT

Dr. Dimitri Theodoratos, Committee Member — Date
Associate Professor, Department of Computer Science, NJIT

Dr. Yi Chen, Committee Member — Date
Professor, School of Management, NJIT

Dr. Moshiur Rahman, Committee Member — Date
Principal Scientist, AT&T

BIOGRAPHICAL SKETCH

Author: Arwa M. Wali

Degree: Doctor of Philosophy

Date: August 2018

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2018
- Master of Science in Information Systems,
  New Jersey Institute of Technology, Newark, NJ, 2011
- Bachelor of Science in Computer Science,
  King Abdulaziz University, Jeddah, Saudi Arabia, 2002

Major: Computer Science

Presentations and Publications:

M.E. Houle, V. Oria and A.M. Wali, "k-LIDoids: A Subspace Clustering Algorithm using Local Intrinsic Dimensionality," In preparation for submission.

M.E. Houle, V. Oria, K.R. Rohloff and A.M. Wali, "LID-Fingerprint: A Local Intrinsic Dimensionality-Based Fingerprinting Method," in *International Conference on Similarity Search and Applications*, 2018.

M.E. Houle, V. Oria and A.M. Wali, in "Improving k-NN Graph Accuracy Using Local Intrinsic Dimensionality," in *International Conference on Similarity Search and Applications.*, Springer, 2017, pp 110-124.

J. Geller, S. Ae Chun and A. Wali, "A Hybrid Approach to Developing a Cyber Security Ontology," in *Proceedings of 3rd International Conference on Data Management Technologies and Applications.* SCITEPRESS-Science and Technology Publications, Lda, 2014, pp 377-384.

A. Wali, S. A. Chun and J. Geller, "A bootstrapping approach for developing a cyber-security ontology using textbook index terms," in *Availability, Reliability, and Security (ARES), 2013 Eighth International Conference on.*, IEEE, 2013, pp. 569–576.

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

((وَمَا تَوْفِيقِي إِلَّا بِاللَّهِ ۚ عَلَيْهِ تَوَكَّلْتُ وَإِلَيْهِ أُنِيبُ)) - هود - الآية:٨٨
"My success can only come from Allah. In Him I trust, and to Him I return." *Quran, Surah Hud, Aya:88*

عَنْ أَبِي هُرَيْرَةَ رضي الله عنه أن رَسُولُ اللَّهِ صَلَّى اللَّهُ عَلَيْهِ وَسَلَّمَ قال: "مَنْ سَلَكَ طَرِيقًا يَلْتَمِسُ فِيهِ عِلْمًا سَهَّلَ اللَّهُ لَهُ بِهِ طَرِيقًا إِلَى الْجَنَّةِ" (رواه مسلم)
Abu Huraira (May Allah be pleased with him) reported: The Messenger of Allah, peace and blessings be upon him, said, "Allah makes the way to Paradise (Jannah) easy for him who treads the path in search of knowledge."
Source: Ṣaḥīḥ Muslim 2699

"اللَّهُمَّ انْفَعْنِي بِمَا عَلَّمْتَنِي، وَعَلِّمْنِي مَا يَنْفَعُنِي، وَزِدْنِي عِلْمًا"
O Allah, benefit me by that which You have taught me, and teach me that which will benefit me, and increase me in knowledge.

To my parents.
The strong and gentle souls who taught me to trust in Allah and believe in hard work. Thanks for supporting and encouraging me to strive for excellence.

To my beloved husband, Emad.
The reason of what I become today, you are my inspiration and my soulmate. Thanks for your love, great support, and continuous care.

To my children, Hammam Lateen Bateel and Azzam.
The real treasure from Allah. Thanks for being patients with me throughout these years.

ACKNOWLEDGMENT

Thanks to Allah Almighty for giving me the strength and ability to understand, learn, and complete this dissertation. It is a great pleasure to acknowledge my deepest thanks and gratitude to my co-advisors, Dr. Vincent Oria and Dr. Michael E. Houle, for suggesting the topics of this research, and for their valuable guidance and advice during the work on this dissertation.

I would like also to extend my thanks and appreciation to Dr. Ali Mili, Dr. Dimitri Theodoratos, Dr. Yi Chen, and Dr. Moshiur Rahaman, for being members in my doctoral dissertation committee. I am very grateful for their time, kindness, encouragement, and valuable comments. I am particularly thankful for the comments of Dr. Ali Mili with respect to the suggested areas of enhancement for writing and editing the dissertation.

I would like also to thank Dr. Cristian Borcea, the (former) chair of the Department of Computer Science (CS) at New Jersey Institute of Technology (NJIT), for opening his door to me, listening to my concerns, and for giving me kind and generous advice and support. I am also very grateful to him for providing me the teaching opportunities in the CS department.

For the initial encouragement to commence doctoral studies, and/or advocacy of my candidacy including the early stage of my doctoral research, I would like to thank Dr. James Geller, Dr. Soon A. Chun, and Dr. Reza Curtmola for their time, commitment, and sincere advice.

My thanks also goes to my sponsors, King Abdulaziz University (KAU) and Saudi Arabia Cultural Mission (SACM) for their great financial support during my graduate studies. I would also like to thank both the CS Department administration at NJIT, particularly Kathy Thompson and Angel J. Butler, for their help in getting all my paperwork done, and the Department of Academic and Research Computing

Systems (ARCS) of the Information Services and Technology (IST) Division at NJIT for providing me with continuous technical assistance. I would also like to thank the National Institute of Informatics (NII) in Japan for providing me the research internship opportunity.

I furthermore want to express my appreciation to my colleagues, Christopher Ochs, Xiang Ji, and Jichao Sun from the CS Department at NJIT, and Oussama Chelly from NII, for their help and technical support. Most important, I would like to thank my parents, my mother, Fatima Alhindi, and my father, Dr. Mahmoud Wali, for their financial support and passionate encouragement. I am extremely grateful to my husband, Dr. Emad Alharbi, and my children, Hammam, Lateen, Bateel, and Azzam, for their patience, continuous help and endless love during this long journey. I am also thankful to my entire family for their praying and support.

Last but not least, I am humbly extend my acknowledgment to all the people who were concerned and co-operated with me in this regard.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 k-NN Graph Construction 3

1.2 Subspace Clustering 5

1.3 Binary Fingerprinting and Indexing 8

2 RELATED WORK 13

2.1 Feature Selection Techniques 13

2.1.1 Supervised Feature Selection 14

2.1.2 Unsupervised Feature Selection 15

2.2 Binary Fingerprinting and Indexing 21

2.2.1 Binary Fingerprinting Generating 21

2.2.2 Algorithms for Fast Search with Binary Data 23

2.2.3 Local Dimensionality Reduction for Indexing 25

2.3 Support-Weighted Local Intrinsic Dimensionality 26

2.3.1 Local Intrinsic Dimensionality 27

2.3.2 Support-Weighted Local Intrinsic Dimensionality Measure . . . 30

3 IMPROVING k-NN GRAPH ACCURACY USING LOCAL INTRINSIC DIMENSIONALITY 33

3.1 Overview of NNF-Descent 33

3.1.1 Local Laplacian Score, Feature Ranking, and Sparsification . . 34

3.1.2 NNF-Descent 34

3.2 Improving NN-Descent Graph with Weighted ID 35

3.2.1 Defining Support-weighted ID (wID) for each Feature 36

3.2.2 NNWID-Descent 37

3.2.3 Variants of NNWID-Descent 39

3.3 Experiments 40

TABLE OF CONTENTS
(Continued)

Chapter **Page**

3.3.1 Datasets 41

3.3.2 Competing Methods 42

3.3.3 Performance Measure 42

3.3.4 Default Parameters 42

3.3.5 Effects of Varying the Sparsification Rate Z 43

3.3.6 Effects of Varying the Neighbor List Size K 49

3.4 Conclusion 50

4 k-LIDoids: A SUBSPACE CLUSTERING ALGORITHM USING LOCAL INTRINSIC DIMENSIONALITY 52

4.1 Preliminaries 54

4.1.1 Notations 54

4.1.2 k-medodis Clustering 54

4.1.3 Defining Support-weighted ID (wID) for each Feature per Cluster 56

4.2 k-LIDoids Algorithm 58

4.2.1 Initialization Phase 59

4.2.2 Iterative Phase 59

4.2.3 Termination Criteria for the Clustering Convergence 62

4.2.4 Time Complexity 64

4.3 Experimental Framework 64

4.3.1 Competing Methods 64

4.3.2 Datasets 65

4.3.3 Parameters Setting 66

4.3.4 Evaluation 67

4.3.5 Comparison Against the Competing Methods with Respect to ARI and the Maximum Number of Iterations M 70

4.3.6 Comparison Against the Competing Methods with Respect to the Expected Measurements 72

TABLE OF CONTENTS
(Continued)

Chapter **Page**

4.3.7 Comparison Against the Competing Methods with Respect to the Clustering Convergence 74

4.3.8 Case Studies for the Clustering Accuracy 76

4.4 Conclusion 82

5 LID-FINGERPRINT: A LOCAL INTRINSIC DIMENSIONALITY-BASED FINGERPRINTING AND INDEXING METHOD FOR SIMILARITY SEARCH 83

5.1 LID-Fingerprint Framework 84

5.1.1 Notations 85

5.1.2 Fingerprinting Process 85

5.1.3 Indexing Process 87

5.1.4 Nearest Neighbor Search Process 92

5.1.5 Updating the Index 94

5.1.6 Information Hiding Aspects of LID-Fingerprint 97

5.2 Experimental Framework 98

5.2.1 Competing Methods 98

5.2.2 Datasets 99

5.2.3 Accuracy of LID-Fingerprint based on Nearest Neighbors Graph Construction vs. Sparsification 99

5.2.4 Comparison of LID-Fingerprint with its competitors 103

5.2.5 Preprocessing Time 111

5.3 Conclusion 112

6 CONCLUSION AND FUTURE WORK 114

Bibliography 117

LIST OF TABLES

Table **Page**

3.1 Average Time in Seconds per Iteration for each Dataset 49

4.1 Datasets used in the Experiments 68

4.2 CLIQUE Parameters Setting for the Density Threshold (ϵ), and the Grid Size ($gridSize$) 68

4.3 k-LIDoids: Confusion Matrix for MAGIC 77

4.4 k-LIDoids: Confusion Matrix for ONP 78

4.5 k-LIDoids: Confusion Matrix for MiniBooNE 79

4.6 k-LIDoids: Confusion Matrix for Wearable Computing 80

4.7 k-LIDoids: Subset of Features per Cluster in MAGIC 81

4.8 k-LIDoids: Subset of Features per Cluster in ONP 81

4.9 k-LIDoids: Subset of Features per Cluster in MiniBooNE 81

4.10 k-LIDoids: Subset of Features per Cluster in Wearable Computing 82

5.1 Parameter Setting for all Datasets: the Number of Iterations M for k-LIDoids, the Number of Iterations T for the Object Fingerprinting Process, the Features Sparsification rate Z, and the Cardinality Values for the Clusters and Objects Fingerprints, $m_I = D - (Ceil(D * Z) * M)$ and $m_L = D - (Ceil(D * Z) * T)$, Receptively 105

5.2 Preprocessing Time (in Seconds) for all Methods Except the Linear Scan Search 112

LIST OF FIGURES

Figure **Page**

1.1 (a) An illustration of K-NN graph with K=3, for a set of six objects in Euclidean space, (b) The NN-Descent's principle. 3

1.2 Subspace clustering illustrated for a 3-D dataset with two clusters,the first cluster defined by axes (features) X and Y, and the second cluster defined by by axes (features) Y and Z. 7

1.3 A possible scenario for searching of encrypted data on a server or the cloud. 11

3.1 Performance of NNWID-Descent and its variants with varying values of Z, and $K = 100$ for Google-23, HAR, ISOLET, MNIST, ALOI, and RLCT datasets. 44

3.2 Performance of NNWID-Descent and its variants with varying values of Z, and $K = 100$ for ONP, Statlog, and Wearable Computing datasets. 45

3.3 Performance of NNWID-Descent and the competing methods with varying values of Z, and $K = 100$ for Google-23, HAR, ISOLET, MNIST, ALOI, and RLCT datasets. 47

3.4 Performance of NNWID-Descent and the competing methods with varying values of Z, and $K = 100$ for ONP, Statlog, and Wearable Computing datasets. 48

3.5 Performance of NNWID-Descent with different values of K and fixed Z ($Z = 4\%$ for RLCT, MNIST, 2% for ALOI, 40% for ONP and Statlog, and 50% for Wearable Computing). 51

4.1 Examples of set L in two clusters, where L is defined as local neighborhood for each cluster representative, and p-NN is local neighbor objects list for each medoid. 58

4.2 Adjusted Rand Indices (ARI) for all methods in comparison with respect to the total value of sparsification rate Z as, 12% for ALOI-100 (Total Sparsified Features =80/641), 83% for Wearable Computing (Total Sparsified Features =15/18), 97% for ONP (Total Sparsified Features =58/60), 70% for MAGIC (Total Sparsified Features =7/10), 96% for MiniBooNE (Total Sparsified Features =48/50), and 96% for SDD (Total Sparsified Features =46/48). 71

4.3 Expected Precision, Recall, and Cosine values for all methods in comparison in a specific iteration (M, Z) as (30, 0.093) for ALOI-100, (9, 0.5) for Wearable Computing, (5,0.71) for MAGIC, (27,0.9) for ONP, (18,0.75) for SDD, and (19,0.76) for MiniBooNE. 73

List of Figures
(Continued)

Figure **Page**

4.4 ARI values for k-LIDoids, PROCLUS, and the Correlation Model with respect to the clustering convergence. k-LIDoids reaches a specific iteration and sparsification rate before terminated (M, Z, $\text{wID}_{average}$), as (200, 0.62, N/A) for ALOI-100, (9, 0.5, 3.89E+36) for Wearable Computing, (5,0.71, 0.310386162) for MAGIC, (24,0.8, 0.223351866) for ONP, (10,0.42, 3.09E+36) for SDD, and (19,0.76, 0.022918872) for MiniBooNE. 75

5.1 Binary fingerprint vectors, CF_1 and CF_2 defined together with clusters C_1 and C_2, respectively. 87

5.2 Multi-level index structure of LID-Fingerprint. 90

5.3 Comparison between NNWID-Descent graph (p=10) and LID-Fingerprint fingerprints graph with different values of p (10, 50, and 100). 102

5.4 The 20-NN search performance, in terms of the average accuracy, distance, and time, among the competing methods. For all datasets, the Min-Hash's time is $> 200ms$. The number of distance evaluations of the linear scan is equal to the size of the datasets. 107

5.5 Comparison between LID-Fingerprint indexing and SUSHI in terms of K-NN search with different values of K (20, 40, 60, 80, and 100). . . . 109

5.6 The average search performance, in terms of time and distance evaluations, for the LID-Fingerprint indexing and SUSHI. For all datasets, the results are averaged over all values of K. The number of distance evaluations of SUSHI for MiniBooNE and Wearable Computing datasets is > 100000. 111

CHAPTER 1

INTRODUCTION

Modern data analysis tools have to deal with a massive amount of digital data, which becomes more easily to acquire and to store. The size of this data is not only measured by the number of samples collected, but also with the number of features that characterize these samples [1]. In novel applications such as multimedia and recommender systems, this data is often represented as object feature vectors in high-dimensional spaces. Dealing with high-dimensional data is always a challenge for the state-of-the-art machine learning and data mining algorithms, because of the so-called "curse of dimensionality". As the dimensionality increases, the discriminative ability of similarity measures between any two objects diminishes to the point where those algorithms that depend on them lose their effectiveness. Therefore, in order to help increasing the learning algorithms performance, it is necessary to considerably reduce the number of features/dimensions. This is where dimensionality reduction techniques come into play, whereas dimensionality reduction can be defined as the process of reducing the number of dimensions in order to obtain a set of the most important dimensions for the dataset objects.

Many researchers in statistics, computer science, and applied mathematics try to develop novel solutions of computational techniques in order to cope with the dimensionality reduction problems [2]. Dimensionality reduction techniques are essential for providing compact object representations, and reducing the storage space and the computational complexity of search and learning tasks. In general, these techniques are rarely performed in isolation. Instead, they are often work as preprocessing steps or integrated with other algorithms such as k-NN graph search and cluster analysis.

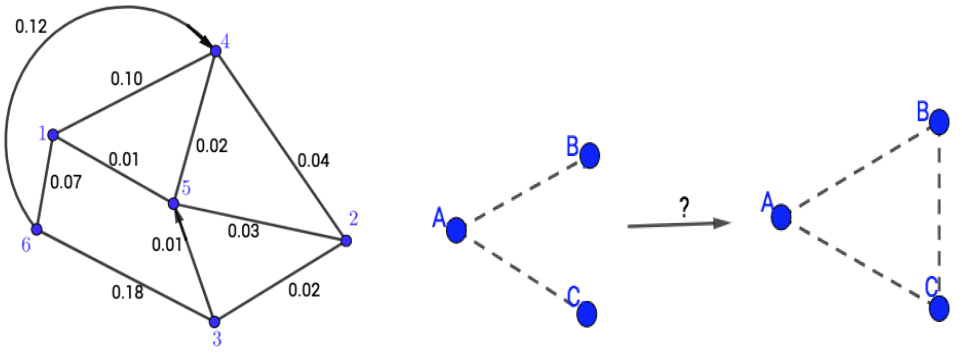
The dimensionality reduction can be done in two different ways: First, feature selection, such as feature weighting [3, 4], that keeps only a subset of most useful features from the original dataset. Second, feature extraction, which tries to construct a lower-dimensional space that captures most of the useful information of the original space. Principal Component Analysis (PCA) [5], which creates linear correlations of the original features, is the most widely used feature extraction technique. Dimensionality reduction techniques can be further divided into (a) global; where all objects in the dataset will be reduced to the same dimensions, and (b) local; where the reduced dimensions are defined locally for one or few related (neighbors) objects.

One of the promising new local feature selection criteria is Support-Weighted Intrinsic Dimensionality (support-weighted ID, or wID) [6]. Support-weighted ID is an extension of the Local Intrinsic Dimensionality (LID) measure introduced in [7, 8], which does not require a construction for the basis dimensions as PCA. Support-weighted ID estimates the contribution of each feature to the overall intrinsic dimensionality. In fact, support-weighted ID measures the ability of each feature to locally discriminate between objects in the dataset.

With the aim to improve the accuracy and performance of k-nearest neighbor (k-NN) search and cluster analysis, this dissertation is mainly concerned with the design and analysis of algorithms based on integration of support-weighted ID within two particular data mining problems, k-NN graph construction and subspace clustering. The compact object and cluster representations from these algorithms are further exploited to define a new binary fingerprinting and indexing framework for the high-dimensional data stored on the cloud.

1.1 k-NN Graph Construction

The k-nearest neighbor (k-NN) graph is a key data structure used in many applications, including machine learning, data mining, and information retrieval. Some prominent examples for k-NN graph utilization include object retrieval [9], data clustering [10], outliers detection [11], manifold ranking [12], and content-based filtering methods for recommender systems [13]. The k-NN graph is usually obtained by connecting each object to its k closest objects in a dataset, where the used distance measure defines the closeness (as shown below in Figure 1.1 (a)) .



(a) K-NN graph (b) The principle of NN-Descent

Figure 1.1 (a) An illustration of K-NN graph with K=3, for a set of six objects in Euclidean space, (b) The NN-Descent's principle.

The construction of k-NN graphs using brute-force techniques requires quadratic time, and is practical only for small datasets [14]. One recent technique that efficiently constructs an approximate k-NN graph in a generic metric space is NN-Descent [14]. NN-Descent is an iterative algorithm that follows a simple transitivity principle (as shown above in Figure 1.1 (b)): two neighbors of a given data object have a higher chance of being neighbors of each other. When ground truth class information is available, the accuracy of a k-NN graph can be measured in terms of the proportion of edges that connect nodes sharing the same class label. A common approach

for maximizing k-NN graph accuracy is to incorporate dimensionality reduction techniques in the graph construction process. This can be done either independently as a preprocessing step using techniques such as Sparse Principal Component Analysis (Sparse PCA) [15], or integrated within the graph construction process itself, such as feature weighting [16] or other supervised feature selection approaches [17]. However, supervised feature selection would depend on ground truth information, which may not be always available.

In [18], an unsupervised method is presented, NNF-Descent, that iteratively and efficiently improves k-NN graph construction using the Local Laplacian Score (LLS) as a feature selection criterion. LLS favors those features that have high global variance among all objects, but less variance among the neighborhood of a given target object. The NNF-Descent method identifies locally noisy features relative to each object in the dataset — that is, those features having larger LLS scores. The noisy features are then gradually modified using a local sparsification process so as to decrease the distances between related objects, and thereby increase k-NN graph accuracy. NNF-Descent has already shown significant improvement in the semantic quality of the graphs produced, and superior performance over its competitors on several image databases [18]. However, NNF-Descent is a conservative method in that only a fixed small number of noisy features are sparsified in each iteration. With greater rates of feature sparsification, the k-NN graph accuracy tends to decrease. This also occurs when increasing the neighborhood size k beyond (roughly) 10. NNF-Descent is designed for datasets with dense feature vectors. In sparse datasets, vectors may contain very few non-zero features, in which case the sparsification process may incorrectly remove valuable features [18].

We address the problem of improving the trade-off between k-NN graph accuracy and the degree of data sparsification by proposing NNWID-Descent. NNWID-Descent is a similarity graph construction method that utilizes the NNF-Descent

framework while integrating support-weighted ID, as a new feature selection criterion, to identify and retain relevant features of each object. Unlike LLS, which is a variance-based measure, support-weighted ID penalizes those features that have lower locally discriminative power as well as higher density. Through extensive experiments on various datasets, we show that NNWID-Descent allows a significant amount of local feature vector sparsification while still preserving a reasonable level of graph accuracy.

1.2 Subspace Clustering

Cluster analysis is a branch of statistics that has a significant contribution in many research areas including businesses intelligence [19], data mining [20], machine learning [21], image pattern recognition [22], Web search [23], and even security [24]. Cluster analysis or clustering is defined as a process of dividing a set of data objects, according to some distance measurement, into multiple meaningful groups (clusters) such that objects are similar within clusters and dissimilar to objects in other clusters [25].

Several clustering algorithms [26, 27, 28, 29] have been proposed to be used as a standalone tool for analyzing a given dataset, or as a preprocessing step for other algorithms such as feature selection and classification. However, traditional clustering techniques may lose their scalability and effectiveness because of (1) the issues associated with the "curse of dimensionality", and (2) the dataset objects may be effectively represented by only a subset of the dimensions or features, which is often much smaller than the actual space [30].

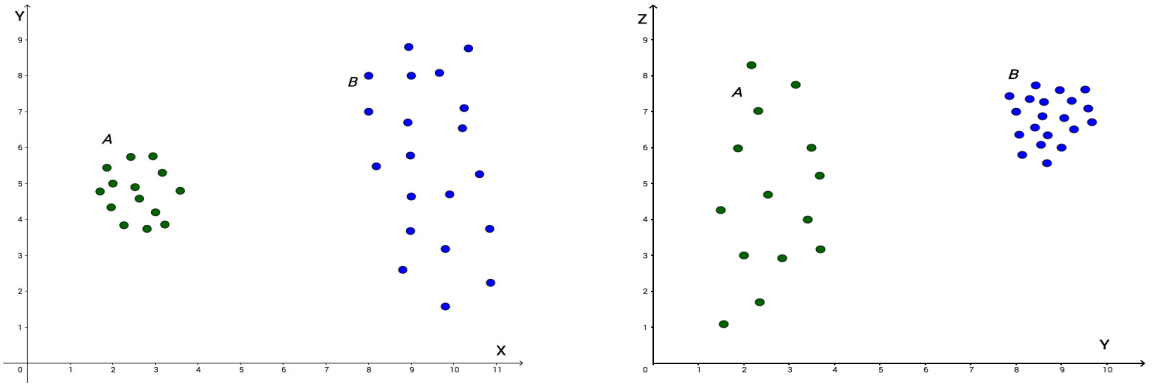
The questions that naturally arise—due to the curse of dimensionality— are: *how to effectively and efficiently find the actual objects that correspond to each cluster of a given datasets?* And *how to define the best minimum subset of dimensions (features) for each cluster without reducing the clustering performance?*. Generally,

clustering the dataset objects using only the relevant features to the clusters will lead to a better clustering performance.

One possible solution for clustering high-dimensional noisy data is by using a sparse clustering framework. This framework aims to cluster objects using a chosen subset of features that is relevant to the entire dataset objects. Dimensionality reduction approaches, in terms of features extraction and feature selection, are employed within the clustering process in the sparse clustering framework in order to (i) improve the clustering quality, and (ii) reduce the storage and computational requirements of high-dimensional datasets. Correlation-based clustering methods using PCA, and Spectral Clustering [31], are examples of sparse clustering algorithms that are based-on features extraction techniques. However, these methods require intensive computation time for large datasets, which negatively impacts their scalability. Conversely, the sparse clustering algorithms that are based on feature selection techniques, such as feature weighting [3, 4] or Lasso-type constraint (L_1-norm) [32], perform clustering on only a small set of useful features. The sparse clustering framework, in general, is important for many applications that use streaming data or text data where many noisy features are present. The sparse clustering algorithms are global, therefore, these algorithms are unusable for revealing useful information from each cluster.

In practice, however, the subset of features that is relevant to one cluster is not necessarily relevant to another cluster. For instance, there are billions of online news articles, where each is described by a collection of keywords. It is necessary for personalized news recommendation systems to be able to group these articles based on a different set of keywords that match specific user preferences [33]. Therefore, there is a need to define a framework that simultaneously finds the clusters of data objects, and determines each subset of features (subspace) that describes each cluster. This framework is known as *subspace clustering* (Figure 1.2). The subspace clustering

framework integrates feature evaluation techniques with clustering algorithms in order to localize the search of the subspace that is relevant to each cluster individually. Several popular subspace clustering algorithms have been proposed in the literature. CLIQUE [34] and MAFIA [35] for example, discover all dense regions of dataset objects in all possible subspaces. Other common algorithms, such as PROCLUS[36] and δ-clusters [37], start with initial approximate clusters, then iteratively evaluate and select features for each cluster in order to regenerate the optimal quality clusters. However, subspace clustering algorithms can lead to a loss of information and a distortion in the resulting clusters. This is due to the fact that these algorithms explicitly and rigidly select small subsets of relevant features and excessively prune away other features in order to define the underline cluster subspaces.



(a) Cluster A defined by features (X, Y)

(b) Cluster B defined by features (Y,Z)

Figure 1.2 Subspace clustering illustrated for a 3-D dataset with two clusters,the first cluster defined by axes (features) X and Y, and the second cluster defined by by axes (features) Y and Z.

In order to avoid the loss of information and the clustering distortion, one possible technique that can be adopted by subspace clustering algorithms is to conservatively and gradually remove the noisy features until obtaining the best subset of informative and important features per object or cluster. Motivated by this technique, we address the subspace clustering issues by presenting a subspace clustering

algorithm called k-LIDoids that extends the utility of the support-weighted ID feature selection criterion within a clustering framework. By using support-weighted ID to identify a relevant subset of features locally for each cluster, k-LIDoids is able to construct clusters together with finding a low-dimensional subspace for each of the clusters. Experimentally, we show that our method can define the minimum subset of features per cluster while maintaining or increasing the clustering accuracy.

1.3 Binary Fingerprinting and Indexing

The increasing number of private multimedia data (documents, images, audio and videos) stored on the cloud on a daily basis requires novel solutions for providing a secure search for these data. Cloud storage, in general, can be abused by different malicious actors such as "Man-in-the-Cloud" (MITC) attacks, data breaches and data loss, or phishing attacks. Passive adversary (a.k.a honest-but-curious attack) where semi-honest people try to see or infer information from the data is a constant threat for the data stored on the cloud.

Information hiding is one of the information security branches that is concerned about the information search and exchange obscuring— either in transit or stored— from unintended observers [38]. Areas related to information hiding include, covert channels [39], steganography [40], anonymity [41, 42, 43], and watermarking [44].

Fingerprinting, which is an application of watermarking, refers to an attempt of creating a unique identification for a data object. Fingerprinting can be active or passive. In active fingerprinting, the unique identification (i.e., serial number) is embedded into a digital object using watermarking techniques. Passive fingerprinting, which is the focus of this work, uses some features of the object to define an identifier for that object [38]. In addition to providing copyright and data privacy, fingerprinting can be used to trace any illegal use of the data [45], or verify the integrity of the data [46]. Typically, fingerprints have much shorter and compact numeric sequences

than the actual objects. One possible representation for the fingerprint is the binary (bits) vector, called *binary fingerprint* or *binary code*, constructed from the object feature vector.

There has been much research investigating the obtaining of compact binary codes in a variety of digital data domains. The state-of-the-art methods are based on Locality Sensitive Hashing (LSH) [47, 48] and have been successfully used to generate audio and images fingerprints [49, 50]. Other methods such as machine learning techniques [51, 52, 53], Spectral Hashing (SH) [54], bag-of-words approach [55], triplet histogram [56], local point aggregates [57], and fingerprint generation [58], have also been considered for fingerprinting purposes.

Binary fingerprinting is also a highly desirable for representing sensitive high-dimensional digital and multimedia data. For a better construction of the fingerprints and in order to avoid the "curse of dimensionality", it is necessary to locally reduce the dimensionality of these high-dimensional objects, and then perform fingerprinting for each object in the reduced space. In large datasets, however, the number of the generated fingerprints can be high and the similarity measure evaluations in a linear scan search can be extensive to compute. Therefore, developing an efficient and effective indexing data structure is important for reducing the number of similarity evaluations and speeding up the search by pruning the search space once the query is presented. In general, the efficiency of the search in the binary fingerprint data depends on the length of each fingerprint and the computational complexity of the used index structure, which has to be accurate, secure, and salable for large databases.

Existing techniques for indexing the binary data such as Semantic Hashing [53] or Locality Sensitive Hashing [59] may lose their efficiency and effectiveness with the increasing number of 0's dimensions in the binary fingerprints. Furthermore, common indexing methods that applied on dimensionality-reduced representations

assume that the dataset is globally reduced, thus, these methods are impractical for indexing binary fingerprints.

Limited existing studies have embedded the local dimensionality reduction techniques in the index construction process for datasets. These studies aim to create a multilevel index format based on using subspace clustering methods. Two main strategies are adopted in the construction of these types of indexes. First, a subspace clustering algorithm is used to generate a single clustering for the whole dataset, then a dimensionality reduction technique, such as PCA, is applied on each cluster separately. A tree-based index is then created for each of the reduced representations by using one or more dimensions of the cluster [60]. Examples of this type of indexing are LDR [61] and MMDR [62]. Second, a hierarchal nesting of subspace clustering framework is used in the index construction process so as to consider multiple representations of the objects according to different dimensions in each level. This strategy applies a subspace clustering algorithm recursively in each level of the tree in order to re-cluster existing subspace clusters. SUSHI [60] and $\triangle^{+}$-tree [63] are examples of such type of indexing methods.

Despite the high performance of the two aforementioned strategies that are adopted in the multilevel indexing, these strategies have two main shortcomings. First, they consider full feature representations for the objects in the leaf level in order to prune the search space; computing the similarity between the query and the objects using the full high-dimensional space will degrade the search efficiency and data privacy. Second, the utilized subspace clustering algorithms lose their performance when excessively eliminating many features from the clusters.

Perhaps one source of difficulty is to automatically generate and index a huge database of fingerprints for high-dimensional data in order to achieve a secure–against any passive adversaries– and efficient search and retrieval. This database could be saved on a network server or the cloud. The potential scenario, shown in Figure 1.3, is

to store an efficient fingerprints' index with its encrypted data objects in the server's database. When presented with any query object, its fingerprint is computed and compared against the stored index. The results of a fingerprints set, with a size K, can be associated with their encrypted data objects and sent to the client. On the client end, the fingerprints part is extracted from the results using a specific extracting algorithm, and the data part is decrypted. The fingerprints are then modified with the actual feature vector values of the data in order to refine the results and obtain a smaller and more accurate results set—with a size $\acute{K} \ll K$.

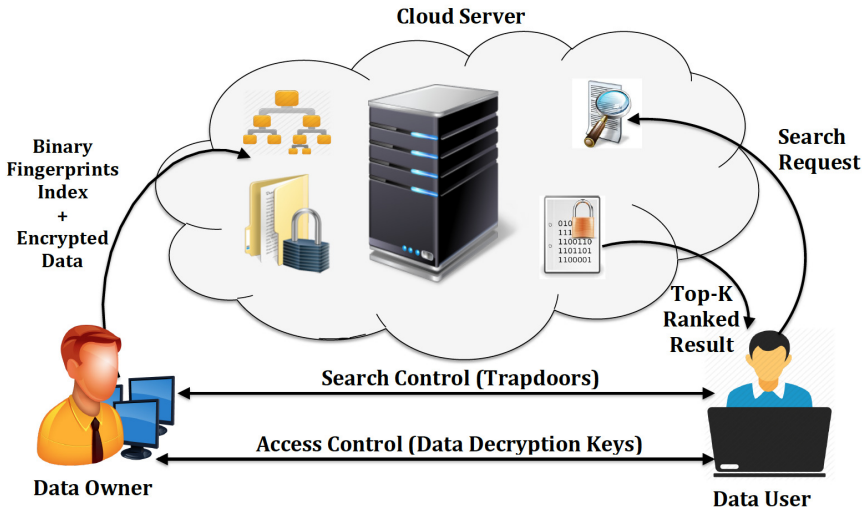


Figure 1.3 A possible scenario for searching of encrypted data on a server or the cloud.

As a possible application for NNWID-Descent and k-LIDoids, we define LID-Fingerprint, a new binary fingerprinting and multi-level indexing framework for the data represented in high-dimensional spaces. LID-Fingerprint can be used for hiding the information on the server side (or the cloud) as a way of preventing passive adversaries. The binary fingerprints are derived from the sparse representations of the data objects, which are resulted from using the feature selection criterion, (support-weighted ID), within a similarity graph construction method,

NNWID-Descent. Furthermore, we define a multi-level index structure based on the subspace clustering algorithm, k-LIDoids, to provide an efficient and secure similarity search for large fingerprint repositories. LID-Fingerprint ensures data suppression and data masking by reducing the overall quality of the data in order to prevent any sensitive information to be inferred. Experiment results have shown that LID-Fingerprint is able to generate compact binary fingerprints, and also provides an efficient and secure indexing technique that allows a reasonable level of search accuracy.

The remainder of this dissertation is organized as follows. In Chapter 2, the related work and the support-weighted ID criterion are discussed. The works of k-NN graph construction and subspace clustering algorithms are presented in Chapters 3 and 4, respectively. Chapter 5 presents the work of the binary fingerprinting and multi-level indexing framework. Finally, the dissertation is concluded in Chapter 6.

CHAPTER 2

RELATED WORK

In this chapter, we discuss the main work related to this dissertation. Section 2.1 surveys the literature on feature selection techniques with more emphasis on those techniques that are integrated with search and clustering algorithms as the basis for the works presented in Chapters 3 and 4. Section 2.2 introduces some of the state-of-the-art subspace binary fingerprinting and Indexing algorithms that motivate the work proposed in Chapter 5. Section 2.3 then presents the feature selection criterion, Support-Weighted Intrinsic Dimensionality (support-weighted ID, or wID) [6] as the basis for the design and analysis of the algorithms presented in this dissertation.

2.1 Feature Selection Techniques

Feature selection can be defined as the process of selecting a subset of relevant features and removing noisy and redundant ones in order to reduce the computation time and improve the learning accuracy [64]. Feature selection techniques, which are widely applied in data mining and machine learning problems such as search and clustering, can be categorized as wrapper-based [65] or filter-based [66]. The wrapper-based techniques select subsets of features using heuristic search strategies, and evaluate the quality of each combination of reduced features using a target learning algorithm. The filter-based techniques, on the other hand, evaluate feature relevance using a statistical measure to assign a score to each feature. The features are then ranked based on the assigned scores and either selected or removed from the dataset. Filter-based techniques are more desirable in the context of k-NN graph construction and clustering analysis because there is no a specific learning algorithm

is required, and the computational time is much lower compared to wrapper-based approaches.

In the subsequent sections, we survey feature selection techniques, which can be embedded in both supervised and unsupervised learning algorithms, with a particular emphasis on unsupervised methods, as the main interest of this dissertation, are considered.

2.1.1 Supervised Feature Selection

Feature selection methods are commonly used in supervised learning (i.e classification) algorithms to maximize their predictive accuracy. The fundamental principle of these methods is using evaluation criteria to measure the relevance or the correlation between the features and the dataset class labels. For example, Song et al. [67] presented a filter method, BAHSIC, that runs a backward selection algorithm that discards features based on their correlation, measured by the Hilbert-Schmidt independence criterion, with the class labels. In [17], a supervised feature selection method was presented that uses an improved k-NN graph-based text representation model to reduce the number of features and predict the category of the text in the test set.

Han et al. [16] proposed a Weight Adjusted K-Nearest Neighbor (WAKNN) classification scheme where the weights of the features are learned in small steps using an iterative algorithm to gradually improves the classification objective function. In [68], the authors use class labels for finding the features, with low variance values, that distinguish each cluster's objects in a modified fuzzy c-means clustering framework.

Rashedi et al. [69] integrated image feature adaptation and selection in a simultaneous process. The authors use a hybrid meta-heuristic swarm intelligence-based search technique, called mixed gravitational search algorithm (MGSA), such that each image database has its own parameters for feature extraction. These parameters

values are encoded together with a binary vector corresponding to the selected features.

Jiang et al. [70] proposed a relevance feedback learning method for online image feature selection. The returned results for a given query image are labeled as 'relevant' or 'irrelevant' by the user. The most related features to the query concept are then selected using a psychological similarity between the two labeled sets.

Relief [66], Fisher score [71], and Information Gain [72] based methods, are among the most representative algorithms of the supervised feature selection model. However, the previously discussed methods yield good learning results for labeled data objects only, and require ground truth input, which is not always available.

2.1.2 Unsupervised Feature Selection

Typically, class information is not available in unsupervised feature selection methods, and thus, it is difficult to decide the importance of a feature — especially when many of the features may be redundant or irrelevant [73]. Most existing unsupervised feature selection approaches are customized to a particular search or clustering algorithm. In general, several articles in the literature attempt to solve the feature selection using clustering techniques as a primary model [74, 75, 76].

Unsupervised feature selection methods can be further classified into global and local methods. In the subsequent paragraphs, we give few examples of global feature selection. Then, we provide some common local feature selection with more emphasize on those clustering algorithms that are closely related to the methods defined in this dissertation.

Global Feature Selection In global feature selection methods, the features are selected based on their relevancy that has been computed globally using the entire dataset. The Laplacian Score (LS) [77] is one of the most popular unsupervised filter-based methods for generic data. LS selects the features to be used for all objects

in the dataset based on their ability to discriminate among object classes. LS favors those features that have high variance on the entire dataset and low variance within local neighborhoods.

In [78], Yang et al. proposed the unsupervised discriminative feature selection (UDFS) algorithm that incorporates discriminative analysis and $L_{2,1}$-norm minimization into a joint framework. Based on the optimization of an objective function, the most discriminative feature subset is selected from the whole feature set in a batch mode. However, this algorithm requires the number of classes as an input, which is often difficult to define, and also has a large time complexity for high-dimensional datasets.

Global feature selection techniques can be integrated within the clustering process to generate what is so called a sparse clustering algorithm. Sparse clustering algorithms aim to find clusters with respect to a small fraction of the features—among the entire dataset—instead of full features set. Dash and Liu [3] addressed the selecting of subset of important features for the whole dataset in order to assist the clustering process. Their method, RANK, consists of two steps: first, it ranks the features using entropy-based ranking measure. Then, it evaluates the features using a scattering invariant criterion function for clustering, in order to select the best subset of features.

In a similar way, Cai et al. in [79] proposed a Multi-Cluster Feature Selection (MCFS) method which uses a two-step strategy to select features according to spectral clustering. The first step is to learn the correlation between features using spectral clustering, then features are selected using spectral regression with L_1-norm regularization in the second step. MCFS shows a better improvement over Laplacian Score (LS).

The authors in [4] presented a method that minimizes the ratio, called generalized Fisher ratio, of the average of intra-cluster to the the average of inter-cluster

by optimizing variable (feature) weights in k-means clustering. Similarly, in [80], the feature-weight learning used in a fuzzy c-means (FCM) clustering algorithm to assign various weights to different features, based on weighted Euclidean distance, in order to improve the clustering performance.

Witten and Tibshirani [32] proposed a general framework for feature selection in sparse clustering. They applied Lasso-type constraint (L_1-norm), as a feature selection method embedded in the clustering process, with focusing on the k-means and hierarchical clustering methods. Lasso-type constraint is applied in the full batch setting of maximizing between cluster distances rather than minimizing the objective function of the clustering algorithms.

Zhang and Lu in [81] proposed a large-scale sparse clustering (LSSC) algorithm based on a two-step optimization strategy. First, obtaining initial clustering results using k-means algorithm. Then, refining the initial results using a spare coding algorithm, which was sped up using the nonlinear approximation and dimension reduction techniques. For more algorithms that are defined as global feature selection, we refer the reader to [82, 83, 84, 76].

Local Feature Selection Local feature selection methods are based on the idea that the discriminative power and the importance of a feature may vary from one neighborhood to another; they aim to select features based on their relevancy to a given neighborhood. Integrating of local feature selection methods with clustering algorithms called *subspace clustering*. Subspace clustering algorithms aim to search for a relevant subset of features locally to each cluster, which allow them to find the best clustering exist in multiple subspaces. A summary of more recent researches in subspace clustering models and algorithms can be found in [85, 86, 87]. In general, there are two types of subspace clustering, bottom-up approaches, and top-down approaches.

Bottom-up approaches These approaches start from low-dimensional subspaces and search for high-dimensional subspaces if there are possible clusters exist. Searching for high-dimensional subspaces can be reduced using different pruning techniques. CLIQUE [34], for example, is a simple bottom-up density and grid-based method that automatically and efficiently indexes high-density cluster subspaces of high-dimensional datasets, and eliminates subspaces with low density objects. CLIQUE is able to discover irregular-shaped clusters, and objects can belong to multiple clusters.

MAFIA [35] is an extension of CLIQUE that uses a density and grid-based method to improve the efficiently and clustering quality. It also goes one step further by allowing parallelism of the clustering process in order to improve the scalability. ENCLUS [88] is similar to CLIQUE except that it uses an entropy measure for the clustering evaluation rather than the density.

SUBCLU [89], an extension of the DBSCAN [90], is a density-based subspace clustering algorithm for detecting clusters in high-dimensional data. SUBCLU is a greedy algorithm that computes all density-connected clusters that are hidden in the subspaces of high-dimensional data. In contrast to CLIQUE and other grid-based approaches, SUBCLU provides a better clustering quality but requires a higher execution time. Modifications of SUBCLU include FIRES [91] and INSCY [92].

A review of other bottom-up approaches including CBF [93], CLTree [94], and DOC [95] can be found in [85, 86].

Top-down approaches In top-down approaches, subspace clustering algorithms start from the full dimensional subspaces, and search iteratively for the low-dimensional subspaces by using multiple iterations to evaluate and select features in the context of each cluster objects, then refine the resulted clusters. For example, Li et al. [96] introduced a localized feature selection algorithm for clustering that is

able to reduce noisy features within individual clusters. Their algorithm computes, adjusts, and normalizes the scatter separability for individual clusters before applying a backward search technique to find the optimal (local) feature subsets for each cluster.

Kim et al. [97] proposed an evolutionary algorithm that is used as a wrapper around clustering algorithm (k-means) to select a subset of features. This algorithm uses multi-objective fitness functions or Pareto optimization for clustering validity criteria, namely: cluster cohesiveness-related to intra-cluster distance; separation between clusters-related to inter-cluster distance; number of clusters; and number of selected features. The fitness of an individual (i.e., a candidate set of selected features) is computed by running a clustering algorithm with the selected features and measuring the corresponding clustering validity criteria.

Friedman and Meulman [98] defined a method for clustering objects on subsets of attributes by computing a weight for each variable (i.e., attribute) in each cluster. They implemented their variable selection criterion in the context of a hierarchical clustering. Mitra et al. [75] introduced an algorithm that partitions the original feature set into clusters based on a k-NN graph principle. To detect and remove redundant features, their algorithm uses a pairwise feature similarity measure, the Maximum Information Compression index, which finds the linear correlation between features in the clusters. This algorithm has a low computational complexity, since it does not involve any search for feature subsets [75]. However, their model may be too restrictive for the real datasets, since correlations among features within clusters may not exist, or may be non-linear when they do exist [99].

Among the popular top-down methods are, PROCLUS [36], FINDIT [100], and δ-clusters [37]. PROCLUS [36] is the first iterative top-down subspace algorithm. Similar to k-medoids clustering algorithm, it first generates k initial cluster representatives from a sample of high-dimensional dataset, then iteratively

refines the clustering by searching for the appropriate subspaces defining each cluster using the local neighborhood to the cluster representatives. In each iteration, the subset of dimensions with average distances smaller than the average distance of all dimensions in the neighborhood to the medoid is determined as the possible subspace for each cluster. Once all subspaces are defined, the clusters are discovered from the subspaces using the distance measures on subsets of dimensions. PROCLUS is able to find the possible outliers cluster as well.

A Variation of PROCLUS is FINDIT [100], which employs additional heuristics to enhance clustering efficiency and accuracy. FINDIT uses a specific distance measure called the Dimension Oriented Distance (DOD). The DOD measure of each cluster representative is used to find the correlated dimensions (i.e., subspace) of each cluster. The clusters are then formed by assigning the objects to cluster representatives based on the subspaces found.

Yang et al. [37] presented a subspace clustering method named δ-clusters that uses a distance measure to capture the coherence manifested by a subset of objects on a subset of dimensions simultaneously. Correlation measures are used to find the coherence of each object or feature to a particular cluster. The algorithm starts with initial cluster representatives and iteratively enhances the clustering quality by randomly swapping attributes and data objects to improve each cluster. This iterative process is terminated when there are no more improvements occurred in the cluster.

However, all the previous popular approaches require a proper tuning for the input parameters, such as the grid size and the density in CLIQUE and MAFIA, the initial number of cluster representatives in RPOCLUS, the minimum distance between two clusters in FINDIT, and the individual cluster size in δ-clusters [86], which are all difficult to determine.

2.2 Binary Fingerprinting and Indexing

In this section, we first discuss the related work of binary fingerprinting with the main focus on the techniques that provide an efficient and secure similarity search. Then, the main search algorithms in binary data also provided as a motivation for the importance of defining a new indexing technique to enhance the search in binary fingerprints. Last, local dimensionality indexing methods are presented as the basis for the indexing technique proposed in Chapter 5.

2.2.1 Binary Fingerprinting Generating

A number of works on designing and generating fingerprints have been proposed in the literature. The common goals for these works are accelerating the nearest neighbor search beside hiding the information from unauthorized observers on network servers or the cloud. Perhaps the Locality Sensitive Hashing (LSH) [101, 102] algorithm is the state-of-the-art method to obtain fingerprints. This algorithm seeks to find an efficient binary representations of high-dimensional data objects by computing hash functions based on random projections. Each random projection contributes few bits in the object fingerprint. The hash functions help in maintaining the similarity between objects in the new binary space [51]. Other LSH-based binary fingerprinting methods are also proposed such as Min-Hash [103], Super-Bit [104], Simhash [48], geometric min-hashing [105], and Spectral Hashing [54]. Although LSH and its variants work efficiently for high-dimensional datasets, it has been reported that when the number of bits is fixed and relatively small, LSH may perform very poorly in generating accurate fingerprints for the data objects [51].

A common step for many fingerprinting methods is to include dimensionally reduction techniques within binary codes generating. For instance, in [51], Torralba et al. proposed a method that adapts machine leaning techniques for dimensionally reduction, such as Boosting [49] and Restricted Boltzmann Machines [53], as well

as LSH in order to convert image Gist descriptors into compact binary codes for large databases. This method allows fast object recognition with an accuracy value comparable with using full descriptors. Strecha et al. [106] used Linear Discriminant Analysis (LDA) as a dimensionality reduction technique for original vectors prior learning the binary codes based on a global matrix projection using a gradient-based method called AdaBoost [107]. Caballero et al. [58] proposed FiG, an automatic active fingerprint generation system. Their system automatically generates candidate queries, sends them to a set of training hosts, identifies useful queries, and applies machine learning techniques, which include dimensionally reduction, to identify a different set of possible fingerprints. However, the previously mentioned methods are supervised that depend on the objects labels information in order to define the binary codes for the datasets.

Unsupervised fingerprinting methods are also defined, where the dataset labels are not required. The authors in [108] introduced two quantization methods to convert real-valued Spectral minutiae features into binary codes called, Spectral Bits, and Phase Bits. They applied two feature reduction techniques, Column Principle Component Analysis (CPCA) and Line Discrete Fourier Transform (LDFT) [109] prior generating the compact fixed-length binary representations for minutiae templates using the quantization methods. The proposed methods mask out (change to 0's) the features that have absolute values are below certain thresholds, while converting other features to 1's. Gong and Lazebnik [110] defined an iterative quantization (ITQ) method for learning the binary codes. The method starts by transforming the data using PCA-binary coding scheme, then an alternating minimization approach is used for refining these transformed data in order to reduce the quantization error.

Farooq et al. [56] presented a technique that satisfy two criteria, anonymity and recoverability. The technique converts already generated fingerprints to anonymous

binary representations based on minutiae triplets that can be used in a template-based matching. These binary representations are then transformed to anonymous representations by assigning a unique key to each user. Basically, anonymous fingerprinting construction requires tow main phases. First, selecting invariant features from the original fingerprint that will be used later for computing the binary fingerprint. Second, the anonymous binary fingerprint is generated by assigning a key to each user. This key helps in randomizing the user template, which can be redefined if this template has been compromised. However, this technique needs to calculate all the possible triples invariant features, and therefore, it has high computational costs.

2.2.2 Algorithms for Fast Search with Binary Data

Binary data, or binary fingerprints, allow a sub-linear and efficient search using binary (bits) operations with respect to the database size. The typical distance metric used in binary data search is the Hamming distance, which can be computed quickly between any two binary vectors as a bitwise XOR operation followed by a set bit count on the result. In this section, we will give a brief review for the main data structures used to perform the nearest neighbor search in binary data. The most naive method is a brute-force linear scan that computes the Hamming distance between the binary representation of the query vector with every fingerprints in the database. This method is only practical for very small datasets.

Salakhutdinov and Hinton [53] presented a nearest-neighbor search for binary data called Semantic Hashing. Each binary vector is corresponded to a memory address such that retrieving similar neighbors to a query vector is performed by retrieving all objects within the Hamming ball around that query vector. This approach is extremely fast, but it is impractical for long binary vectors since the Hamming distance between objects becomes large, which increases the number of objects within the Hamming ball that becomes difficult to explore [111].

Locality Sensitive Hashing (LSH) [101] is a popular randomized hashing framework for a fast approximate nearest-neighbor search. LSH inserts database objects into several hash tables such that similar objects are assigned the same hash key, and hashed to the same buckets with a high probability. The hash keys usually are low-dimensional binary (bits) vectors, where these bits are generated using several hash functions. Given a query object, it is directly hashed to the stored buckets, and its matched buckets are retrieved, which their elements then are compared based on a brute force matching using the Hamming distance. Many LSH-based algorithms have been developed in order to improve the accuracy and speed of the original one [112, 47, 48, 113, 105]. However, designing appropriate hash functions may limits the flexibility of these algorithms.

Hierarchical clustering is another popular technique used for the search in binary data. Brin [114] proposed Geometric Near-Neighbor Access Tree (GNAT). GNAT does a hierarchical decomposition of the search space, where some of data objects are used to represent the cluster centers instead of computing the cluster means. This change helps in speeding up the search and allows the tree to work in any metric space including the Hamming space. The authors in [115] extended GNAT by building multiple hierarchical cluster trees for the binary vectors. To search for the nearest neighbor objects for the query object, all trees are traversed simultaneously in a best-first approach. This method is implemented in an open source library called Fast Library for Approximate Nearest Neighbors (FLANN) [116]. However, even with the excellent performance of the existing hierarchical clustering techniques to search for binary data, these techniques assume that the binary data is already generated, which is different than the scope and the objective of the work presented in Chapter 5 of this dissertation.

2.2.3 Local Dimensionality Reduction for Indexing

In order to handle queries in high-dimensional datasets, two main strategies of dimensionality reduction are used within indexing methods: global dimensionality reduction, and local dimensionality reduction. In a global dimensionality reduction strategy, all dataset objects are reduced to the same space. The index is then created only based on this reduced space. Examples of these indexing methods include TV-Tree [117] and iDistance [118]. However, the global dimensionality reduction strategy does not consider the local correlations between the objects in the dataset.

Local dimensionality reduction strategy, on the other hand, considers the local correlations in the dataset by reducing the dimensions of each object or group of objects individually. We provide here some common techniques in Local dimensionality reduction as they are closely related to the work in this dissertation (Chapter 5). For example, LDR [61], is a multilevel index structure that uses a subspace clustering to divide the whole dataset into local correlated clusters. After performing dimensionality reduction in each cluster individually, a separate high-dimensional index is constructed for every subspace. MMDR [62] differs from LDR in that it constructs a single index for every reduced dimension in every cluster. Both LDR and MMDR use PCA as a dimensionality reduction method but they do not consider multiple representations for objects in different dimensions [60].

In [63], Cui et al. introduced $\triangle^+$-tree a multilevel index structure based on a hierarchal top-down subspace clustering. In this index, the dataset is partitioned into clusters and then each cluster is divided further into sub-clusters in each inner level of the tree. In each level, the data objects are represented using different dimensionalities based on applying PCA. The number of dimensions increases toward the leaf level of the index, and the full feature representations of the objects are used in the leaf level. The lower dimensions of $\triangle^+$-tree in each level can help in pruning the search

space and reducing the computational cost of distance evaluations between queries and dataset objects.

Günnemann et al. [60] introduced SUSHI, a more general framework for a multilevel index structure based on a hierarchal nesting top-down subspace clustering. Similar to $\triangle^+$-tree, SUSHI provides a multi-representation of objects based on the reduced dimensions in each level. However, instead of increasing the number of dimensions toward the leaf level, the number of dimensions for each cluster in each level is determined by the subspace clustering algorithm used. To prune the search space, SUSHI uses a compact description of each cluster, called Subspace Enclosing Rectangular (SER), so that the queries traverse through different filters of the index. Exposing a subset or a full set of dimensions in the index levels of both $\triangle^+$-tree or SUSHI leads to insecure search either on the network server or the cloud.

2.3 Support-Weighted Local Intrinsic Dimensionality

The intrinsic dimensionality of a dataset can be defined as the minimum number of dimensions/features needed to represent the data without information loss [119]. Generally, a dataset, X, with a number of dimensions m, have Intrinsic Dimensionality (ID) equals to d, if its objects lie entirely within d-dimensional subspace (where $d < m$). There are many ID estimation measures that have been proposed in the literature: classical measures, which includes the Hausdorff dimension, Minkowski-Bouligand or "box counting" dimension, and the correlation dimension; fractal-based measures of the space filing capacity or self-similarity of the data [120, 121]; and topological or local measures where ID is estimated based on using the neighborhood of each dataset sample, such as Near Neighbor Algorithm [122], and the methods based on Topological Representing Networks (TRN) [123]. Popular projection techniques, such as linear and nonlinear PCA [124, 125, 126], can also produce as a byproduct an estimate of the ID of the datasets.

In the theory of intrinsic dimensionality, the expansion-based models (such as the minimum neighbor distance (MiND) [127]), expansion dimension (ED) [128], and the generalized expansion dimension (GED) [129]), quantify the ID in the vicinity of a point of interest in the data domain, by measuring the rate of growth in the number of data points encountered as the distance from the reference sample increases. As a motivating example, the volume of an m-dimensional ball, in Euclidean space, grows proportionally to r^m, when its size is scaled by a factor of r. If we consider the volumes V_1 and V_2 are defined for two balls of differing radii r_1 and r_2, respectively, and centered at a common reference point, then, the expansion dimension m from the ratios of the volumes and the distances from this reference point can be deduced as follows:

$$\frac{V_2}{V_1} = \left(\frac{r_2}{r_1}\right)^m \Rightarrow m = \frac{\ln\left(V_2/V_1\right)}{\ln\left(r_2/r_1\right)} \tag{2.1}$$

For finite datasets, GED formulations are obtained by estimating the volume of balls as the numbers of points they enclose [129]. Since classical expansion models estimation is restricted to a neighborhood around the sample of interest, then they can provide a local view of the dimensional structure of the data by treating the probability mass as a proxy for the volume.

2.3.1 Local Intrinsic Dimensionality

Instead of regarding intrinsic dimensionality as a characteristic of a collection of data points distances from a supplied reference point, the GED was recently transfered to a statistical setting of continuous distance distributions, of a random variable $\mathbf{X}$. By letting the radii r_1 and r_2 of the two balls (defined above) be $r_1 = x$ and $r_2 = (1+\epsilon)x$, and $\epsilon \to 0^+$, then, ID can be modeled as a function of distance $\mathbf{X} = x$. This leads to a formal definition of the local intrinsic dimensionality LID [130].

Definition 1 (Local Intrinsic Dimensionality (LID) [130, 131]) *Let* $\mathbf{X} > 0$ *be a random variable denoting the distance* x *from a given reference point to other data samples. If the cumulative distribution function* $F(x)$ *of* $\mathbf{X}$ *is positive and continuously differentiable at distance* $x > 0$*, the LID of* F *at distance* x *is defined by:*

$$\mathrm{ID}_F(x) \triangleq \lim_{\epsilon \to 0} \frac{\ln(F((1+\epsilon).x)/F(x))}{\ln(1+\epsilon)} \triangleq \frac{r \cdot F'(x)}{F(x)} \tag{2.2}$$

whenever the limit exists.

The last equality in Equation 2.2 follows by applying l'Hôpital's rule to the limit [130]. The notation $F(x)$ of the probability measure is analogous to the volume V in Equation 2.1; however, the underlying distance measure need not be Euclidean. Under the distributional interpretation, the original dataset defines a sample of distances at a given point. The intrinsic dimensionality (referred as 'Local ID' or LID) of this distance distribution F is estimated. The definition of LID at x can be extended to be defined as the limit, when the radius x tends to zero ($x \to 0^+$), whenever this limit exists:

$$\mathrm{ID}^*_F(x) \triangleq \lim_{x \to 0^+} \mathrm{ID}_F(x) \tag{2.3}$$

The cumulative distance function $F(x)$ in the relative rate, which is described by ID_F, increases as the distance x increases from 0. Thus, ID_F can be estimated using the distances of x to its k nearest neighbors within the sample [8]. In the ideal case, ID_F equals the dimension of the submanifold when the data in the vicinity of x is distributed uniformly within a submanifold. However, in general these distributions are not ideal, the manifold model of data does not perfectly apply, and ID_F is not an integer. Nevertheless, the local intrinsic dimensionality does give a rough indication of the dimension of the submanifold containing x that would best fit the data distribution in the vicinity of x. For more details regarding to the LID model, the readers may refer to [130, 6].

Estimation of LID The smallest k nearest neighbor distances from a given point can be regarded as 'extreme events' associated with the lower tail of the underlying distance distribution. Therefore, the modeling of neighborhood distance values can be investigated from the viewpoint of extreme value theory (EVT) (a branch of statistics). It is shown, in [132], that the EVT representation of the cumulative distribution F completely determines function ID_F, and that the EVT index is in fact identical to ID_F^*

Under very reasonable assumptions, the tails of continuous probability distributions converge to the Generalized Pareto Distribution (GPD), a form of power law distribution [133]. From this, Amsaleg et al. [8] developed several estimators of LID to heuristically approximate the true underlying distance distribution by a transformed GPD. Among these, the Maximum Likelihood Estimator (MLE), which has a relative stability and convergence properties, exhibited a useful trade-off between statistical efficiency and complexity. Given a reference sample $x \sim P$, where P represents the data distribution, the MLE estimator of the LID at x is defined as follows:

$$\mathrm{ID}_F(x) = -\left(\frac{1}{k}\sum_{i=1}^{k}\ln\frac{x_i}{w}\right)^{-1}. \tag{2.4}$$

where x_1, ... , x_k are observations of a random distance variable $\mathbf{X}$ taking values in the range $(0, w]$. Each x_i denotes the distance between x and its i-th nearest neighbor within a sample of points drawn from P, and $x_k = w$ is the maximum of the neighbor distances (k-nearest neighbor distance). In practice, the sample set is drawn uniformly from the available training data (omitting x itself), which itself is presumed to have been randomly drawn from P. We emphasize that the LID defined in Equation 2.3 is a theoretical quantity, and that LID as defined in Equation 2.4 is its estimate. In the remainder of this dissertation , we will refer to Equation 2.4 to calculate the LID estimates.

2.3.2 Support-Weighted Local Intrinsic Dimensionality Measure

We propose in this dissertation a new feature evaluation strategy based on the Local Intrinsic Dimension ('Local ID', or 'LID') model originally appearing in [7] and discussed above. To recall, given a distribution of distances with a univariate cumulative distribution function F that is positive and continuously differentiable in the vicinity of distance value x, the indiscriminability of F at x is given by

$$\mathrm{ID}_F(x) \triangleq \frac{x \cdot F'(x)}{F(x)}\,. \tag{2.5}$$

The indiscriminability reflects the growth rate of the cumulative distance function at x; it can be regarded as a probability density associated with the neighborhood of radius x (that is, $F'(x)$), normalized by the cumulative density of the neighborhood (that is, $F(x)/x$). The local intrinsic dimension has been shown to be equivalent to a notion of local intrinsic dimensionality, which can be defined as the limit $\mathrm{ID}_F^* = \lim_{x \to 0^+} \mathrm{ID}_F(x)$. However, the notion of local ID as proposed in [130, 6] is considerably more general, in that the original model of [7] has been extended to handle multivariate real-valued functions that are not necessarily the cumulative distribution functions of distance distributions.

When considering a distance distribution on a space of many features, it is natural to ask which variables or features are contributing most to the overall discriminability of the function or cumulative distribution function (as the case may be). Two variables or features with the same local ID value may not necessarily have the same impact on the overall ID value. To illustrate this, let Φ and Ψ be the respective cumulative distribution functions of two univariate distance distributions on distance variable x.

The indiscriminability $\mathrm{ID}_\Phi(x)$ can be thought of as having a 'support' equal to the probability measure associated with distance x — namely, $\Phi(x)$; similarly, the support for $\mathrm{ID}_\Psi(x)$ would be $\Psi(x)$. Even when the indiscriminabilities $\mathrm{ID}_\Phi(x)$ and

$\mathrm{ID}_\Psi(x)$ are equal, if (say) the support $\Phi(x)$ greatly exceeded $\Psi(x)$, one would be forced to conclude that the features associated with ID_Φ are more significant than those of ID_Ψ, at least within the neighborhood of radius x.

For the comparison of the discriminabilities of different features in our proposed algorithms in this dissertation, we will adopt the following support-Weighted ID complexity measure. This measure has the highly desirable theoretical advantage of being additive across features (for more details we refer the reader to [6]).

Definition 2 (Support-Weighted ID [6]) *Let F be a real-valued multivariate function over a normed vector space $(\mathbb{R}^m, \|\cdot\|)$, and let $\mathbf{x} \neq \mathbf{0} \in \mathbb{R}^m$ be a vector of positive norm. The* support-weighted indiscriminability of F at $\mathbf{x}$ *is defined as*

$$\mathrm{wID}_F(\mathbf{x}) \triangleq F(\mathbf{x})\,\mathrm{ID}_F(\mathbf{x}) = \mathbf{x}\cdot\nabla F(\mathbf{x})\,. \tag{2.6}$$

Estimating support-weighted ID for the purpose of assessing indiscriminability can be complicated by the need to standardize the distance within which the indiscriminabilities are measured — in a k-NN graph, each neighborhood is associated with its own potentially-unique k-NN distance. If each feature were to assessed at widely-varying distances, there would be no basis for the fair comparison of feature performance.

In practice, however, estimation of ID requires samples that are the result of a k-nearest neighbor query on the underlying dataset. Across such samples, standardization can be achieved using the local ID representation theorem:

Theorem 1 (Local ID Representation Theorem [130]) *Let $\Phi : \mathbb{R} \to \mathbb{R}$ be a real-valued function, and let $v \in \mathbb{R}$ be a value for which $\mathrm{ID}_\Phi(v)$ exists. Let x and w be values for which x/w and $\Phi(x)/\Phi(w)$ are both positive. If Φ is non-zero and*

continuously differentiable everywhere in the interval $[\min\{x,w\}, \max\{x,w\}]$*, then*

$$\frac{\Phi(x)}{\Phi(w)} = \left(\frac{x}{w}\right)^{\mathrm{ID}_\Phi(v)} \cdot G_{\Phi,v,w}(x), \text{ where} \tag{2.7}$$

$$G_{\Phi,v,w}(x) \triangleq \exp\left(\int_x^w \frac{\mathrm{ID}_\Phi(v) - \mathrm{ID}_\Phi(t)}{t}\,\mathrm{d}t\right), \tag{2.8}$$

whenever the integral exists.

For a univariate cumulative distribution function Φ at distance x, we can use Theorem 1 with $v = 0$ to relate the support $\Phi(x)$ with the support at another desired distance w. If n is the size of the dataset that we are given, we choose the distance at which over n selection trials one would expect k samples to fall within the neighborhood — that is, w would satisfy $\Phi(w) = k/n$. The support-weighted ID would thus be:

$$\mathrm{wID}_\Phi(x) = \Phi(x)\,\mathrm{ID}_\Phi(x) = \frac{k\,\mathrm{ID}_\Phi(x)}{n} \cdot \left(\frac{x}{w}\right)^{\mathrm{ID}^*_\Phi} \cdot G_{\Phi,0,w}(x)\,. \tag{2.9}$$

In [130] it is shown that (under certain mild assumptions) the function $G_{\Phi,0,w}(x)$ tends to 1 as $x, w \to 0$ (or equivalently, as $n \to \infty$); also, $\mathrm{ID}_\Phi(x)$ would tend to ID^*_Φ, for which reliable estimators are known [8, 134]. Thus, for reasonably large dataset sizes, we could use the following approximation:

$$\mathrm{wID}_\Phi(x) \approx \frac{k\,\mathrm{ID}^*_\Phi}{n} \cdot \left(\frac{x}{w}\right)^{\mathrm{ID}^*_\Phi}. \tag{2.10}$$

CHAPTER 3

IMPROVING k-NN GRAPH ACCURACY USING LOCAL INTRINSIC DIMENSIONALITY

In this chapter, we address the problem of improving the tradeoff between k-NN graph accuracy and the degree of data sparsification. We propose *NNWID-Descent*, a similarity graph construction method that utilizes the NNF-Descent framework, while integrating the feature selection criterion, Support-Weighted Intrinsic Dimensionality (support-weighted ID) (see Chapter 2 Section 2.3.2), to identify and retain relevant features of each object. Support-weighted ID penalizes those features that have lower locally discriminative power as well as higher density.

The remainder of this chapter is organized as follows. Section 3.1 provides an overview of the NNF-Descent framework. We outline the proposed NNWID-Descent method in Section 3.2. In Section 3.3, the performance of our method — with experimental results and analysis on several real datasets — is compared to NNF-Descent and other competing methods from the literature. Finally, we conclude the chapter in Section 3.4.

3.1 Overview of NNF-Descent

As the basis for the work presented in this chapter, in this section we provide an overview of the NNF-Descent algorithm [18]. We also describe its feature selection criterion, the Local Laplacian score LLS, and discuss its utilization in feature ranking and sparsification processes.

3.1.1 Local Laplacian Score, Feature Ranking, and Sparsification

Local Laplacian Score LLS is used for feature ranking with respect to individual data object. Assume we have a dataset X with n data objects, each represented by a D-dimensional feature vector $\mathbf{f} = (f_1, f_2, \ldots, f_D)$. We further assume that the vectors are normalized. Then, for an object $x_i \in X$, the LLS score for each of its feature f_i can be computed using the following formula:

$$LLS(f_i) = \sum_j \frac{(f_i - f_j)^2 S_{ij}}{var(\mathbf{f})} \tag{3.1}$$

where $var(\mathbf{f})$ is the variance of feature $\mathbf{f}$, and S_{ij} is the (Gaussian) RBF kernel similarity between two object vectors x_i and x_j defined as:

$$S_{ij} = \begin{cases} exp(\frac{-\|x_i - x_j\|^2}{2\sigma^2}), & \text{if } i \text{ and } j \text{ are connected;} \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Here, σ is a bandwidth parameter. S_{ij} favors neighboring objects x_i and x_j that are likely to share the same class label. A smaller value for LLS(f_i) indicates that the feature is stable among the neighbors of object x_i. The features are ranked for each object in decreasing order of their LLS values, and the top-ranked proportion Z of the ranked list is deemed to be noise. In the sparsification process, the impact of noisy features is minimized by changing their values in the feature vectors to the global mean, which is zero due to normalization.

3.1.2 NNF-Descent

The NNF-Descent framework interleaves k-NN graph construction method using NN-Descent [14] with a feature ranking and sparsification process. Algorithm 1 gives the complete algorithm for NNF-Descent. After normalizing the original vectors of the dataset X, the algorithm starts by computing the initial approximate k-NN graph using NN-Descent [14] (lines 1-2). The NN-Descent procedure depends on the

so-called *local join* operation. Given a target point p, the local join operation checks whether any neighbor of p's neighbors is closer to p than any point currently in its neighbor list, and also whether pairs of neighbors of p can likewise improve each other's tentative neighbor list. Noisy features are gradually identified using LLS, ranked, and then sparsified.

Algorithm 1: NNF-Descent

Input : Dataset X, distance function dist, neighborhood size k, sparsification rate Z, number of iterations T

Output: k-NN graph G

Normalize the original feature vectors of X;
Run NN-Descent(X, dist, k) to convergence to obtain an initial k-NN graph G;
repeat
    Generate a list L of all data points of X in random order;
    foreach *data point* $p \in L$ **do**
        Rank the features of p in descending order of their LLS scores, as computed over the current k-NN list of p;
        Change the value of the top-ranked Z-proportion of features to 0;
        Recompute the distances from p to its k-NN and RNN points;
        Re-sort the k-NN lists of p and its RNNs;
        For each pair (q, r) of points from the k-NN list and RNN list of p, compute $\text{dist}(q, r)$;
        Use $(q, \text{dist}(q, r))$ to update the k-NN list of r, and use $(r, \text{dist}(q, r))$ to update the k-NN list of q;
    end
until *maximum number of iterations* T *is reached*;
Return G

3.2 Improving NN-Descent Graph with Weighted ID

The NNF-Descent framework, which integrates feature ranking and sparsification with k-NN graph construction, serves as the basis for the method presented in this chapter, NNWID-Descent. In NNWID-Descent, instead of feature variance, a measure of the discriminability of features is used for feature ranking. In this section,

we first provide a brief overview of this measure of discriminability, Support-Weighted Local Intrinsic Dimensionality or support-weighted ID (Section 2.3.2). The utilization of support-weighted ID as a feature selection criterion is then presented in Section 3.2.1. Finally, the details of the proposed NNWID-Descent algorithm is given in Section 3.2.2.

3.2.1 Defining Support-weighted ID (wID) for each Feature

Let $X = \{x_1, x_2, x_3, \ldots, x_n\}$ be a dataset consisting of n objects such that each object x_i is represented as a feature vector in $\mathbb{R}^D$. The set of features is denoted as $F = \{1, 2, \ldots, D\}$ such that $j \in F$ is the j-th feature in the vector representation. $K \geq k$ is the neighborhood size for each object per feature. Since the factor k/n in Equation 2.10 can be regarded as constant, the support-weighted ID criterion for feature f_j of object x_i can be simplified:

$$\mathrm{wID}_i(f_j) = \mathrm{ID}_{f_j} \cdot \left(\frac{a}{w_{f_j}}\right)^{\mathrm{ID}_{f_j}} \tag{3.3}$$

where ID_{f_j} is the local intrinsic dimensional estimate for the neighborhood, and w_{f_j} is the distance to the k-th nearest neighbor with respect to feature f_j, respectively. a is any positive constant representing the distance value x. For simplicity, a can be set as an average of total averages of K-NN distances for every feature f_j across a sample of many objects. Equation (3.3) helps to find the most discriminative features by considering both the density of neighborhood around each object and the complexity of local ID with respect to a particular feature f_j.

For feature ranking, a straightforward method is used for selecting the most local discriminative features for each object using wID_i, in which the D features are ranked in descending order of $\mathrm{wID}_i(f_j)$, and a proportion Z of the top-ranked features are determined as candidates for sparsification. Assuming that the feature vectors

have been normalized, the sparsification process (described in Section 3.1.1) will set the values of the least important features to 0.

3.2.2 NNWID-Descent

Algorithm 2 shows how NNWID-Descent proceeds. The input parameters are K, k Z, and T, where $k \leq K$ is the working neighborhood size during the construction of the output k-NN graph, K is the working neighborhood used for computing the wID value for each object's feature, Z is a fixed proportion of features that are sparsified in each iteration, and T is the total number of desired iterations. The feature sparsification rate Z should be relatively small.

The algorithm has two phases: an initialization phase, and a sparsification and refinement phase. In the initialization phase, the algorithm computes a k-NN graph using NN-Descent after normalizing the original vectors of the dataset X (lines 2-3). This step is crucial, since a neighborhood of reasonably high quality is needed for the subsequent refinement phase to be effective.

In line 4, the value of a is precomputed for use in calculating wID values, during the sparsification and refinement phase. As will be described in Section 3.3.4, the value a can be computed as the average of k-NN distances of all features, over a sample of the data objects. The k-NN graph entries are then improved using the sparsification and refinement phase (Lines 6-16). This phase includes three steps: feature ranking, sparsfication, and graph updating. In lines 9-10, the features are ranked in decreasing order according to the wID values obtained from the set of K-NN distances determined by each feature alone. For each object p, the top Z-proportion of features are then sparsified (line 11). As will be described in Section 3.3.4, the value Z is chosen depending on the density of the dataset X. As in [18], only non-zero features are candidates for sparsification, since features with value 0 do not provide discriminative information in the vicinity of p, and thus do not affect the quality of the

Algorithm 2: NNWID-Descent

Input : Dataset X, distance function dist, neighborhood size k for the graph, neighborhood size K for computing wID scores, sparsification rate Z, number of iterations T

Output: k-NN graph G

1 {Initialization Phase}
2 Normalize the original feature vectors of X;
3 Run NN-Descent(X, dist, k) to convergence to obtain an initial k-NN graph G;
4 Set the value of a to the average of k-NN distances computed for all features over a sample of objects.
5 {Sparsification and Refinement Phase}
6 **repeat**
7 Generate a list L of all data points of X in random order;
8 **foreach** *data point* $p \in L$ **do**
9 For each feature, compute the wID (Equation 3.3) using K-NN distances of p per feature with respect to X;
10 Rank the features of p in descending order of their wID scores;
11 Change the value of the top-ranked Z-proportion of features to 0;
12 Recompute the distances from p to its k-NN and RNN points;
13 Re-sort the k-NN lists of p and its RNNs;
14 For each pair (q, r) of points from the k-NN list and RNN list of p, compute $\text{dist}(q, r)$;
15 Use $(q, \text{dist}(q, r))$ to update the k-NN list of r, and use $(r, \text{dist}(q, r))$ to update the k-NN list of q;
16 **end**
17 **until** *maximum number of iterations* T *is reached*;
18 Return G

k-NN graph. Ignoring zero features will ensure that once sparsified, a feature will not be evaluated again in subsequent iterations. Sparsifying a feature vector for p in one iteration will more likely change the nearest neighbors for each feature of p; for this reason, to determine the correct wID value in subsequent iterations, recomputation of the K-NN distances is required for each feature.

Lines 12-15 correspond to Lines 8-11 in NNF-Descent (Algorithm 1) which identify the local join operation and graph update step to improve the graph accuracy.

In the implementation, we set $k \leq K$ to be the length for both RNN and NN lists used in graph updating step.

The time complexity of NNWID-Descent can be divided according to its phases as follows: For the initialization phase, data normalization and NN-Descent —in terms of distance computation until convergence— take $O(Dn)$ and $O(k^2Dn)$ time, respectively. Computing the values of a using average of K-NN distances takes $O(Dn^2)$. For each iteration of the sparsification and refinement phase, feature ranking and selection using wID takes $O(\mathrm{K}Dn + D\log D)$ per object, with total time in $O(\mathrm{K}Dn^2 + Dn\log D)$ over all objects. As with NN-Descent, assuming that the lengths of the RNN lists are in $O(k)$, each iteration of NNWID-Descent takes $O(k^2Dn)$ time for the neighbor list update step. However, the optimizations that have been defined for NN-Descent in [14] can also applied for NNWID-Descent to speed up the local join operation and update steps.

3.2.3 Variants of NNWID-Descent

In this section, we present variants of *NNWID-Descent* that will also be evaluated in the experimentation.

Two-Levels-NNWID-Descent To show the importance of having a good quality neighborhood for obtaining accurate ID estimation, and thus correct wID value per feature, one heuristic solution is to compute wID values once using the neighbors of each object per feature computed from the original features set. Formally, we present a variation of Algorithm 2, *Two-Levels-NNWID-Descent*, by moving lines 9 and 10 to the initialization step. Since the neighbors of each object per feature will not be recomputed in each iteration to find the correct wID value after sparsification process, the time complexity of feature ranking and sparsification steps will reduced to $(D\log D)$ per object, with total time in $O(Dn\log D)$ over all objects.

NNWID-variant As an alternative of computing the wID using the current neighbors for each object recomputed for each feature per iteration, another heuristic solution is to compute the wID by using the current updated neighbors of each object computed in k-NN updating step in *NNWID-Descent* algorithm (lines 11-15). More Formally, we create a variant, *NNWID-variant*, form Algorithm 2 by removing line 9 and modifying line 4 and 10 to:

- Estimate the value of a, which can be set as the average of current *k-NN* distances computed for each object by *NN-Descent* using all or a sample of objects.
- Rank p's features in descending order based on their wID scores computed from p' current k-NN distances;

Unlike *NNWID-Descent*, the time complexity will decrease as computing the values of a using average k-NN distances will take $O(Dn)$ (line 4), and the feature ranking and selection performed using wID will take $O(Dk + D \log D)$ per object with total $O(Dnk + Dn \log D)$ for all objects (line 10).

NNID-Descent In order to illustrate the effect of the weight parameter $(\frac{a}{w_{f_j}})^{\mathrm{ID}_{f_j}}$ for feature ranking, we also contrast *NNWID-Descent* against another variant, *NNID-Descent*, of Algorithm 2. *NNID-Descent* ranks the features according to computing the local Intrinsic Dimensionality ID_{f_j} values without the weight parameter in Equation (3.3). The goal here is to see if the ID_{f_j} value alone will improve the results.

3.3 Experiments

For the comparison of NNWID-Descent with its variants and other competing methods, we conducted experiments to study the influence on performance of varying the feature sparsification rate Z and the working neighbor list size K.

3.3.1 Datasets

Nine real datasets of varying sizes and densities were considered, of which five are image sets:

- **The Amsterdam Library of Object Images (ALOI)** [135] contains 110,250 images of 1000 small objects. Each image is described by a 641-dimensional feature vector based on color and texture histograms.

- **The MNIST dataset** [136] contains 70,000 images of handwritten digits. Each image is represented by 784 gray-scale texture values.

- **Google-23** [137] contains 6,686 faces extracted from images of 23 celebrities. The dimension of the face descriptors is 1,937.

- **The Isolated Letter Speech Recognition dataset (ISOLET)** [138] contains 7797 objects generated by having 150 subjects speak the name of each letter of the alphabet twice. Each object is described by 617 features, and were scaled so that all values lie in the interval $[-1.0, 1.0]$.

- **The Human Activity Recognition Using Smartphones dataset (HAR)** [139] contains 10,299 instances of accelerometer and gyroscope data from 30 subjects performing 6 different activities. Each instance is represented by a feature vector of 561 time and frequency domain variables.

- **The Relative Location of CT dataset (RLCT)** [138] contains 53500 axial CT slice images from 74 different patients. Each CT slice is described by two histograms in polar space. The feature vectors of the images are of 385 dimensions.

- **Wearable Computing**: Classification of Body Postures and Movements (PUC-Rio) dataset [140] contains 165,633 samples collected on eight hours of activities of four healthy subjects in different static postures and dynamic movements. Each sample's features vector has 18 attributes that represent user data such as name, gender, age, height, weight, body mass, and sensor axis values. There are five possible positions (sitting-down, standing-up, standing, walking, and sitting).

- **Online News Popularity dataset (ONP) [141]** contains 39,644 articles' textual extracted data, each has 60 attributes (58 predictive attributes, and 2 non-predictive) that describe different article aspects. The articles are binary classified as popular and unpopular using a decision threshold of 1400 social interactions.

- **Statlog (Shuttle) dataset** [138] from NASA contains 9 continuous numerical attributes related to the positions of radiators in the space shuttle. The dataset has 58,000 instances that divided into 7 classes.

3.3.2 Competing Methods

The performance of NNWID-Descent is contrasted with that of 3 competitors:

- **NNF-Descent**: uses LLS criterion for feature ranking and sparsification (as described in Section 3.1).

- **Random**: as per NNF-Descent, except that for each object the features to be sparsified are selected randomly. The rationale for the comparison with this method is to establish a baseline for the performance of the feature ranking and sparsification criterion.

- **Sparse PCA**: is similar to wID in such that it takes into account the dataset sparsity. In this method, the feature extraction and graph construction are conducted as two separate processes. To allow a fair comparison with other methods, after choosing the highest principal components, an exact k-NN graph is computed (at a computation cost of $O(Dn^2)$).

3.3.3 Performance Measure

We use the graph accuracy as a performance measure. The class labels of the data objects were used to measure the quality of the resulting k-NN graph at every iteration. The accuracy of the resulting k-NN graph is evaluated, as in [18], using the following formula:

$$graph\ accuracy = \frac{\#correct\ neighbors}{\#data \times k}, \tag{3.4}$$

where the 'correct' neighbors share the same label as the query object.

3.3.4 Default Parameters

Except for the case of Sparse PCA, the feature vectors were normalized within each dataset in each experiment performed, and the Euclidean (L_2) distance was employed. In NNWID-Descent and its variant Two-Levels-NNWID-Descent, for all datasets, the value of a in the weight parameter of Equation (3.3) is set to be the average of k-NN distances over a random sample of 100 objects. Furthermore, for all features, the value a is precomputed in advance using the original feature vectors without sparsification.

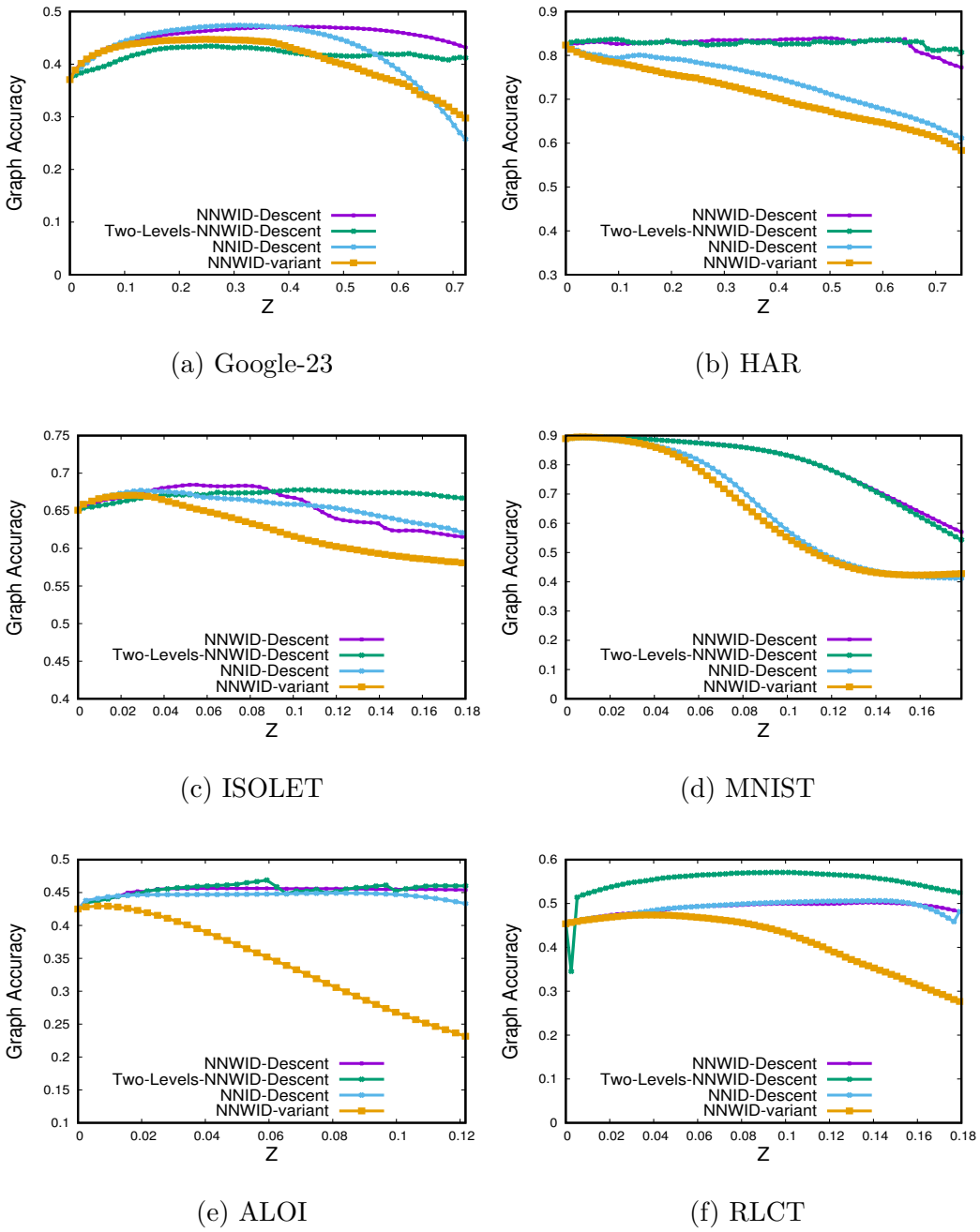
For simplicity, the number of neighbors (K) used for computing wID is set to be equal to the input parameter k, which is also used for computing LLS.

3.3.5 Effects of Varying the Sparsification Rate Z

Parameter Setting In this experiment, we tested the effect on performance of varying Z while keeping K fixed. The choices of Z is varied with different datasets as it depends heavily on the density of the feature vectors. For example, in each iteration, smaller choice of Z ($= 0.0025\%$) for the sparse datasets (MNIST, ALOI, ISOLET, and RLCT) was required to produce gradual changes in graph accuracy with acceptable performance. On the other hand, the dense datasets require a larger starting point to produce perceptible changes in performance from iteration to iteration. For example, Z is set to ($= 0.01\%$) for Google-23 and HAR, ($= 0.055\%$) for Wearable Computing, ($= 0.02\%$) for ONP, and ($= 0.1\%$) for Statlog. The total number of iterations T is set to 70 for all datasets, except for ALOI, Wearable Computing, ONP, and Statlog, for which T is set to 40, 15, 20, and 7, respectively. For all methods in the comparison, the value of K is fixed at 100.

Comparison Against the Variants with Respect to Graph Accuracy To show the importance of selecting a good quality neighborhood as well as the weight parameter in order to find the wID scores for every feature, we compare *NNWID-Descent* with its variants. In contrast to NNWID-Deacent and Two-Levels-NNWID-Descent as described in 3.3.4, the value of a in NNWID-variant is set to be the average k-NN distances computed per feature using the neighbors form the initial constructed K-NN graph by NN-Descent. Figures 3.1 and 3.2 plots the graph accuracy in each iteration for all the variants across a range of Z values.

Results and Analysis For most of the datasets, NNWID-Descent achieves consistently better performance in graph accuracy than all other variants. However,



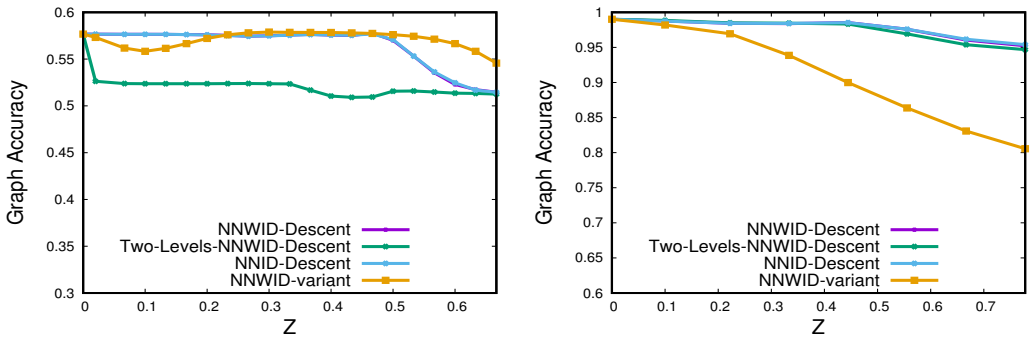
(a) Google-23 (b) HAR

(c) ISOLET (d) MNIST

(e) ALOI (f) RLCT

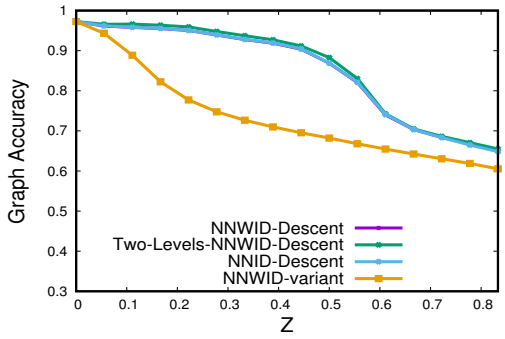
Figure 3.1 Performance of NNWID-Descent and its variants with varying values of Z, and $K = 100$ for Google-23, HAR, ISOLET, MNIST, ALOI, and RLCT datasets.

except for Google-23, and ONP, we noticed that Two-Level-NNWID-Descent gives a very similar or higher graph accuracy than NNWID-Descent, which indicates that the wID scores can be computed initially using the original feature sets since the



(a) ONP

(b) Statlog



(c) Wearable Computing

Figure 3.2 Performance of NNWID-Descent and its variants with varying values of Z, and $K = 100$ for ONP, Statlog, and Wearable Computing datasets.

neighborhood generally have a good quality. On the other hand, NNID-Descent also shows a comperable performance to NNWID-Descent which indicates that the ID_{f_j} alone–without weight parameter–can lead to a good accuracy in some datasets.

Despite of using the weight parameter in NNWID-variant, it gives less performance mostly compared to the other NNWID-Descent variants. The reason is that the value of the weight parameter, $(\frac{a}{w_{f_i}})^{\mathrm{ID}_{f_i}}$, is sensitive to an error correlated with the quality of the intrinsic dimensionality (ID) estimator used; having inaccurate ID estimator leads to instability for the weight parameter values and thus weak feature selection criterion. It is important to realize that obtaining accurate estimates of wID requires that the neighborhood be of generally good quality. In NNWID-Descent, the

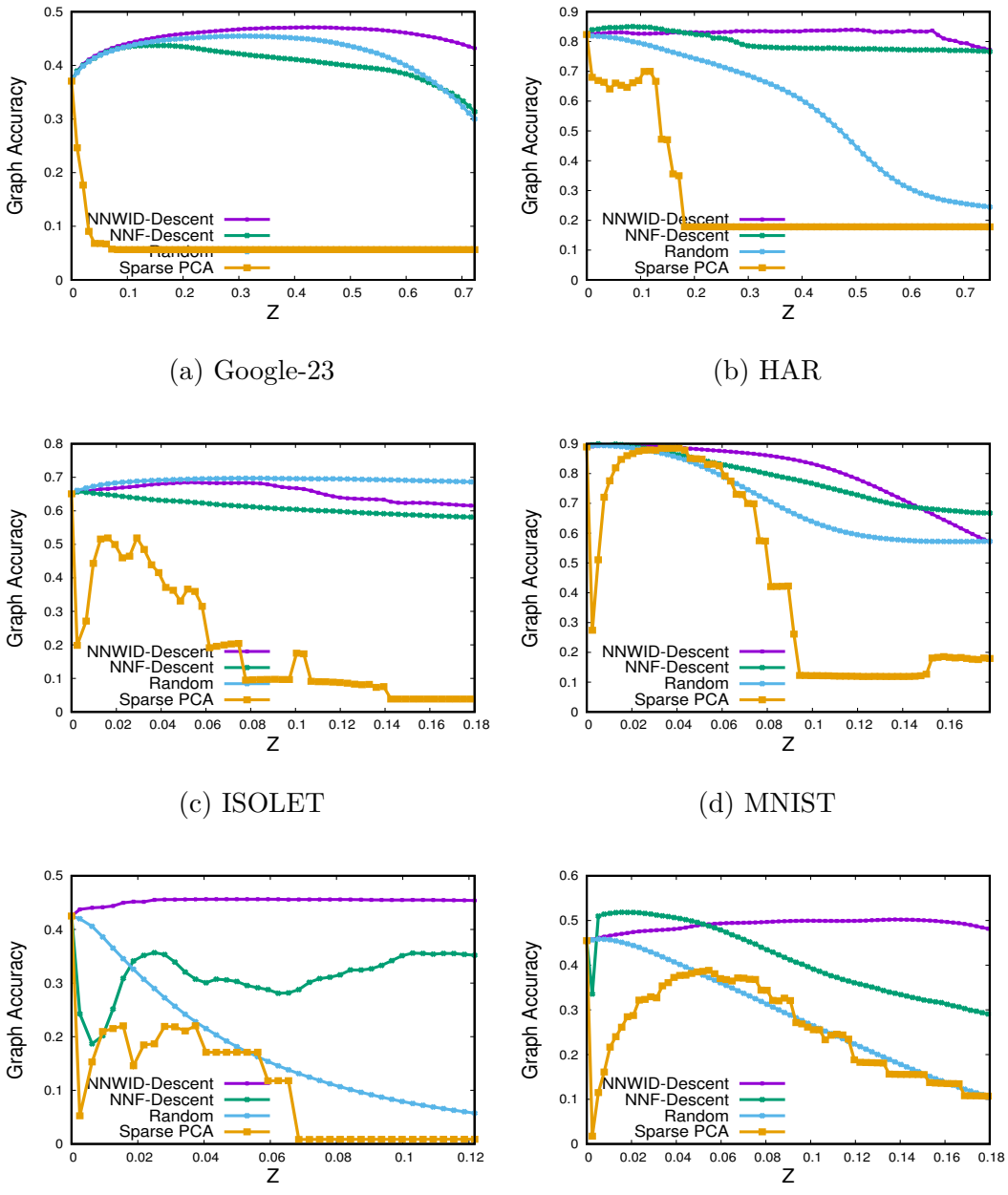
re-computation of neighborhoods after sparsification at each iteration is essential to the quality of wID estimation. However, using distance values computed from the current K-NN graph may lead to less accurate ID estimation if the initial graph has a low quality.

Comparison Against the Competitors with Respect to Graph Accuracy Figures 3.3 and 3.4 show plots of the graph accuracy in each iteration for all the methods, across a range of Z values. For Sparse PCA, the parameter controlling sparsity was set to Z, and the number of principle components selected were set to $D - Z$.

Results and Analysis On six of the nine datasets, compared with its competitors, NNWID-Descent achieves consistent improvements for graph accuracy and resistance to performance degradation as sparsification increases — for ISOLET, it is outperformed only by Random, while in Wearable Computing it is outperformed by NNF-Descent. For the MNIST and Statlog datasets, Sparse PCA has a performance comparable to that of NNWID-Descent for small sparsification rates. NNF-Descent also show a performance similar to NNWID-Descent for Statlog dataset.

Execution Time Except Sparse PCA method as it has different execution strategy, the cost of sparsification and refinement dominates the overall computational performance of all methods that employ this strategy. For these methods, the execution time for the sparsification and refinement phase is displayed in Table 3.1. The displayed times account for the operations of feature ranking, sparsification, and updating of neighbor lists. The table shows the average running time in seconds per iteration for all datasets under consideration.

Since the time for sparsification and neighbor list updating is expected to be the same for all three methods, the observed differences in execution time related



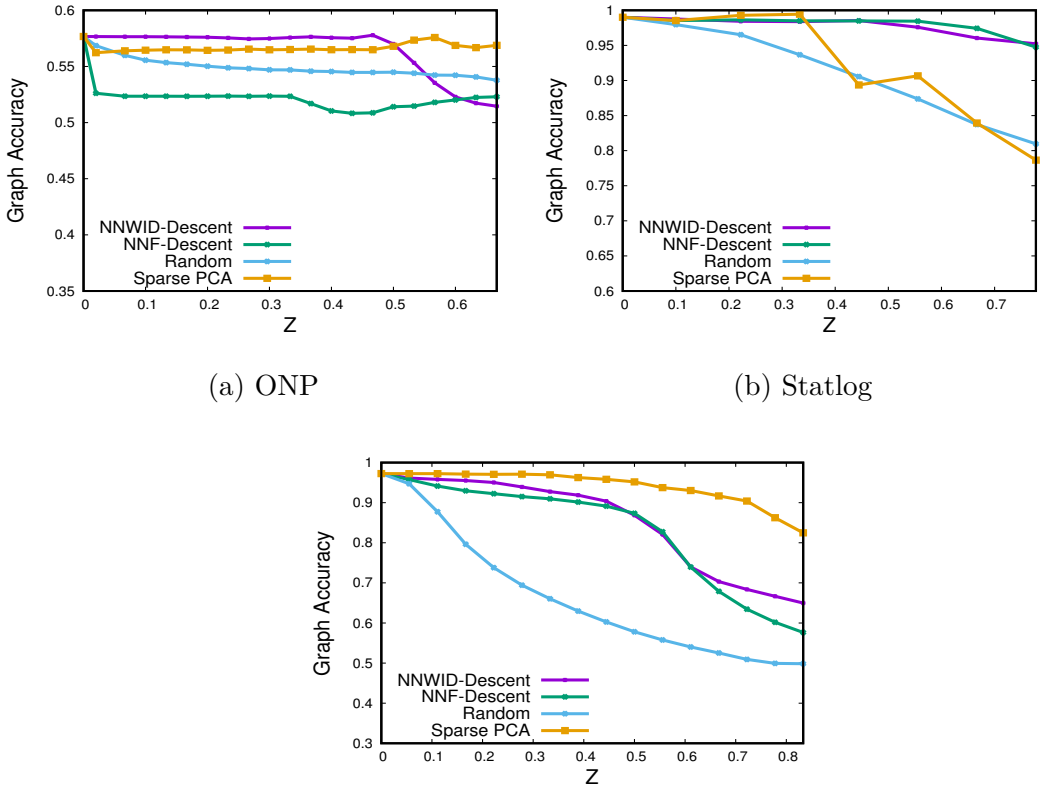
(a) Google-23 (b) HAR

(c) ISOLET (d) MNIST

(e) ALOI (f) RLCT

Figure 3.3 Performance of NNWID-Descent and the competing methods with varying values of Z, and $K = 100$ for Google-23, HAR, ISOLET, MNIST, ALOI, and RLCT datasets.

to differences in the costs of the feature ranking step. As can be observed from Table 3.1, NNWID-Descent has the highest execution cost. This is due to the necessity of computing neighborhood distances for each object per feature in each iteration.



(a) ONP

(b) Statlog

(c) Wearable Computing

Figure 3.4 Performance of NNWID-Descent and the competing methods with varying values of Z, and $K = 100$ for ONP, Statlog, and Wearable Computing datasets.

Despite its larger running time relative to its competitors, NNWID-Descent shows a better potential for the improvement of graph accuracy, and better resistance to performance degradation as sparsification increases. Two-Level-NNWID-Descent and NNWID-variant are more efficient than other methods except the Random method.

In Two-Level-NNWID-Descent, the time complexity reduced significantly since the wID scores are computed initially using the original features and not included in the feature ranking step. On the other hand, considering the current k-NN graph neighbors as in NNWID-variant for computing the wID scores per iteration has less time complexity comparing to recomputing the neighborhood per feature (i.e., NNWID-Descent, NNID-Descent), or using LLS creterian as in NNF-Descent.

Table 3.1 Average Time in Seconds per Iteration for each Dataset

Dataset	*NNWID-Descent*	*Two-Level-NNWID-Descent*	*NNWID-variant*	*NNID-Descent*	*NNF-Descent*	*Random*
Google-23	1431.56	59.25	101.90	944.77	320.96	70.77
ISOLET	1152.43	59.84	74.34	673.36	204.92	73.34
HAR	1275.44	105.87	109.044	832.60	248.75	141.46
MNIST	8281.03	907.16	3733.17	11635.55	5274.55	4429.77
ALOI	55363.56	8454.36	9019.37	56271.92	13053.55	11183.65
RLCT	9549.33	2149.88	1999.10	10026.29	3125.65	3873.19
Wearable Computing	74868.12	18129.39	19764.95	73372.45	19493.05	27922.98
ONP	23444.42	1297.87	1529.69	23422.07	1070.54	2404.91
Statlog	8408.95	2395.50	2655.02	8341.59	2498.0	4378.10

3.3.6 Effects of Varying the Neighbor List Size K

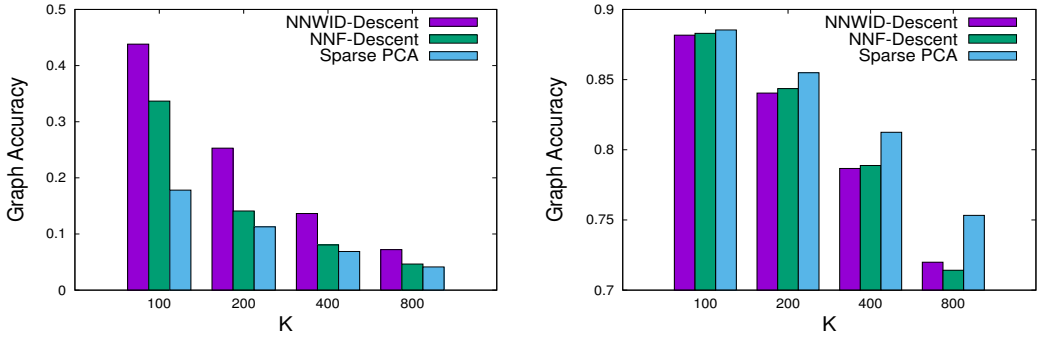
Parameter Setting In this experiment, we compare the performance of NNWID-Descent against NNF-Descent and Sparse PCA as the neighbor list size increases beyond $K = 100$. We show the results for all datasets, except Google-23, HAR, and ISOLET, as the values of K are too large relative to the size of the these datasets. Concretely, K is set to 100, 200, 400, and 800, and Z is fixed at 4% for MNIST and RLCT, 40% for Wearable Computing, ONP, and Statlog, and at 2% for ALOI. These Z values represent approximately the peak graph accuracy achieved in Figures 3.3, and 3.4. The performances across these choices of K are plotted in Figure 3.5.

Results and Analysis We note that for all datasets in comparison, NNWID-Descent still provides better accuracy than other methods as the neighborhood list size K is increased. With MNIST and Wearable Computing, Sparse PCA outperforms other methods as K increases, which indicates that this method can lead to a reasonable graph accuracy for a some datasets with a specific Z rate.. With increasing K, Sparse PCA and NNF-Descent show a comparable performance for NNWID-Descent in both ONP, and Statlog, repectivley.

For all methods, the performance degrades as K increases. In addition, we observe that the relative performances of all methods shown when varying K (Figure 3.5) is still consistent with the performances observed when varying Z (Figures 3.3 and 3.4).

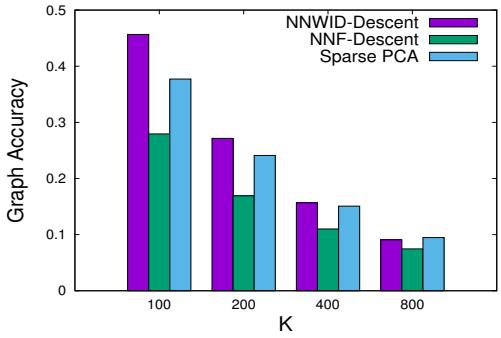
3.4 Conclusion

In this chapter, we presented the NNWID-Descent similarity graph construction method, which utilizes the NNF-Descent framework with a new unsupervised feature selection criterion. This method aimed to improve or maintain k-NN graph accuracy while achieving a significant amount of sparsification of object feature vectors. We proposed the use of support-weighted ID (wID) to identify relevant features with higher discriminative power local to each object. NNWID-Descent ranks the features according to their wID values, then sparsifies those features achieving the smallest values. With respect to the correctness of k-NN graph produced using nine real datasets, NNWID-Descent has been shown to generally outperform its closest competitors, NNF-Descent and Sparse PCA.

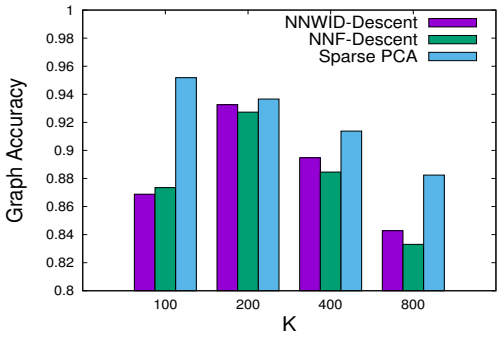


(a) ALOI

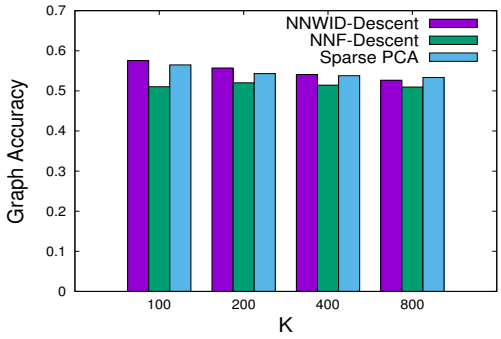
(b) MNIST



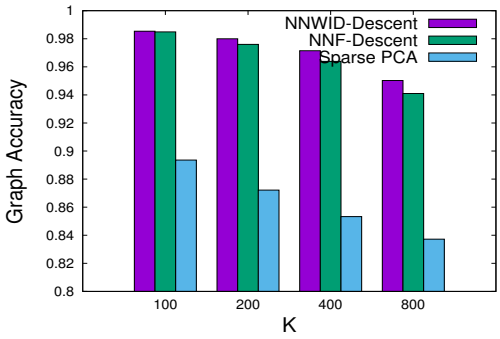
(c) RLCT



(d) Wearable Computing



(e) ONP



(f) Statlog

Figure 3.5 Performance of NNWID-Descent with different values of K and fixed Z ($Z = 4\%$ for RLCT, MNIST, 2% for ALOI, 40% for ONP and Statlog, and 50% for Wearable Computing).

CHAPTER 4

k-LIDoids: A SUBSPACE CLUSTERING ALGORITHM USING LOCAL INTRINSIC DIMENSIONALITY

In this chapter, we address the subspace clustering issues discussed in Chapter 1, such as the loss of information resulted from rigidly selecting small subsets of features to define the underline cluster subspaces, by presenting a new subspace clustering algorithm called k-LIDoids. k-LIDoids extends the utility of the support-weighted ID feature selection criterion within a clustering framework. k-LIDoids simultaneously captures a subset of objects (cluster) on a subset of features (subspace). The algorithm interleaves the process of features elimination (sparsfication) based-on the support-weighted ID criterion (see Chapter 2 Section 2.3.2) with the clustering process in order to iteratively enhance the clustering quality. This enhancement is performed by gradually identifying higher locally discriminative features in the context of each cluster individually.

To define the local closeness of cluster objects in the subspace defined for that cluster, the support-weighted ID criterion can be used to identify those features that have lower locally discriminative power as well as higher density per object in the cluster. Estimating the support-weighted ID criterion for each feature requires computing the local neighborhood objects for each object in the whole dataset. Since it is important to eliminate the features gradually from each object to achieve the local improvement, then the local neighborhood objects need to be updated consistently in order to accommodate the change of the subset of features defined for that object. However, frequently updating the local neighborhood for all dataset objects is computationally expensive. Therefore, for efficiency, it is important to use a clustering method that allows us to limit the number of objects that we are considering for defining the cluster subspaces.

Among all clustering methods in the literature, we select k-medoids clustering algorithm as a basic framework for our proposed method. The k-medoids algorithm employs actual objects—from the dataset—called *medoids* to represent the cluster centers. Using k-medoids has the following two main characteristics: First, using the medoids is desirable for defining the cluster subspace based-on the support-weighted ID criterion because the local neighborhood objects for those medoids are easy to obtained. Second, as an iterative method, k-medoids supports interleaving the process forming of the clustering results with the process of defining the cluster subspaces using the support-weighted ID criterion, which also requires to change the local neighborhood objects of the cluster representatives in order to accommodate the update in the cluster subspaces.

With the integration of the support-weighted ID criterion within k-medoids framework, k-LIDoids often computes k non-overlapping hyper-spherical compact clusters based not only on the dataset objects but also on the feature subspaces defined for each cluster. k-LIDoids has the following advantages over previous subspace clustering approaches.

- Using the supported weighted ID criterion, k-LIDoids is able to measure the ability of each feature to locally discriminate between clusters in the dataset, and therefore, defining the local intrinsic dimensions per cluster.
- k-LIDoids can locally determines each clustering subspace in order to improve the clustering quality without scanning all objects in the dataset. Thus, it is easy to be parallelized to ensure the efficiency and the scalability.

The rest of this chapter is organized as follows. In Section 4.1, we present the preliminaries. Section 4.2 describes our proposed algorithm. Section 4.3 presents the experimental framework for our algorithm with a comparison of the quality of the clustering results with three other well-known algorithms. Finally, we conclude the chapter in Section 4.4.

4.1 Preliminaries

In this section, we first describe the k-medoids clustering that is used as the basic framework for the subspace clustering algorithm proposed in this chapter. Then, we show how we define support-weighted ID for each feature per cluster.

4.1.1 Notations

Let $X = \{x_1, x_2, x_3, \ldots, x_n\}$ be a dataset consisting of n objects such that each object x_i is represented as a feature vector in $\mathbb{R}^D$. Let the set of features is denoted as F=$\{1, 2, \ldots, D\}$ such that $j \in F$ is the j-th feature in the vector representation, and D is the total number of features. p is neighborhood size for each object $x_i \in X$ in the p-nearest neighbor graph G. Suppose that these n objects should be partitioned, according to some distance measurement, into $k(k < n)$ clusters, which is assumed to be given. The set of clusters is denoted as C, where each cluster C_c, $1 \leq c \leq k$, has a cluster representative m_c. The Euclidean (L_2) distance is employed in this work although other distance measures can be adopted similarly. Given a pair of objects x_i, and $x_r \in \mathbb{R}^D$, recall the definition of Euclidean distance:

$$d(x_i, x_r) = (\sum_{j=1}^{D} |x_{ij} - x_{rj}|^2)^{\frac{1}{2}} \tag{4.1}$$

4.1.2 k-medodis Clustering

The k-medoids clustering algorithm searches for an optimal set of k objects (i.e., medoids) from the dataset, which results in the best possible clustering. In general, k-medoids is used to find the local minimum of the following objective function:

$$OF = \sum_{c=1}^{k} \sum_{x_i \in C_c} (d(x_i, m_c))^2 \tag{4.2}$$

where each medoid m_c is an actual object that minimizes the total distance to other objects in the cluster. As we will describe in Section 4.1.3, using medoids is desirable for the feature selection criterion, support-weighted ID, that needs actual objects

from the dataset in order to select important features per object. k-medoids also is less sensitive to outliers; the objects that are far away from the majority of the data. Usually, outliers distort the actual cluster means in algorithms such as k-means clustering.

Generally, the k-medoids clustering algorithm includes three steps: select initial medoids, update medoids, and assign objects to medoids. In the first step, the initial medoids are selected randomly among all objects in the dataset. Then, each object $x_i \in X$ is assigned to a cluster C_c with the closest medoid m_c to that object, and the sum of the total distances between objects and medoids is computed. The algorithm updates each cluster's medoid by searching for a new representative m_c from the entire dataset. In the last step, each object is re-assigned to the nearest medoid to obtain the clustering results. The algorithm executes the second and the third steps iteratively until convergence; when the object assignments no longer change.

Among several k-medoids algorithm variants in the literature [28, 142], we choose the algorithm presented in [143] as the main framework for our method. This algorithm is a local heuristic that runs like k-means clustering algorithm, and it requires to compute the distance between every pairs only once. This k-medoids algorithm also shows, experimentally, an efficiency and effectiveness in handling large datasets. From different medoids initialization methods that are tested by the authors, we use the random initialization in order to avoid both, the increasing in the complexity time, and the bias introduced by selecting the k initial medoids intelligently (i.e., the most middle objects in the dataset). When updating the medoids in the second step (update medoids), this later k-medoids algorithm limits the search for a new representative m_c among the objects within the cluster C_c instead of the entire dataset. However, when the size of any cluster increases, finding the appropriate medoids in each iteration will further increases the overall clustering time complexity.

For efficiency, the second step (update medoids) of the k-meodids algorithm– described above– can be simplified by using a simple heuristic indexing technique that depends on an intuitive idea presented by [144]. This idea shows the relation between k-means and k-medoids clustering; the medoid of a set of objects is often close to the mean. That is, if we have a cluster C_c, the object $x_i \in C_c$, with the smallest average distance to all other objects in C_c, is often close to the mean μ of C_c. This idea is proven in [144] to be true for one cluster taken in isolation from the dataset (k=1), but it is not proven to be true when considering multiple clusters at the same time (k >1). From a practical perspective, however, this simple heuristic indexing technique can still be used when the cluster size becomes relatively large (i.e., $|C_c| \geq \alpha$, i.e., α=10,000). To clarify, searching for a new medoid of each cluster in the second step— of the above k-medoids clustering algorithm—can be simplified by searching among the closet β objects (i.e., β=100) to the cluster mean. We will refer to the adapted k-medoids clustering algorithm [143] with including the above heuristic indexing technique as km-medoids. Algorithm 3, illustrated in the following page, shows the pseudocode of how km-medoids clustering proceeds. It is also important to mention that, for increasing the efficiency, the second step in Algorithm 3 can be executed in parallel for all clusters in the same time.

4.1.3 Defining Support-weighted ID (wID) for each Feature per Cluster

Since the factor p/n in Equation 2.10 can be regarded as constant, the support-weighted ID criterion for feature f_j of object x_i can be simplified:

$$\mathrm{wID}_i(f_j) = \mathrm{ID}_{f_j} \cdot \left(\frac{t}{w_{f_j}}\right)^{\mathrm{ID}_{f_j}} \tag{4.3}$$

where ID_{f_j} is the local intrinsic dimensional estimate for the neighborhood, and w_{f_j} is the distance to the p-th nearest neighbor object with respect to feature f_j, respectively.

Algorithm 3: km-medoids Clustering Algorithm

Input : Dataset X, the number of clusters k

Output: Clusters C

//Step 1: Select initial Medoids

Select k medoids objects, $\{m_1, m_2, \ldots, m_k\}$ randomly from X;

C=Obtain the initial cluster result by assigning each object x_i to the nearest medoids;

$d_{current}$=Calculate the sum of distances from all objects to the their medoids;

repeat

  Set $d_{previous}$=$d_{current}$;

  //Step 2: Update Medoids

  foreach *cluster* $C_c \in C$ *(executes in parallel)* **do**

    if $|C_c| \geq \alpha$ **then** Compute the mean μ of cluster C_c;

    Search for a new medoid among the closest β objects to μ, such that the new medoid is the object that minimizes the total distance to other objects in the cluster; ;

    else Search a new medoid among the objects in the cluster C_c , which is the object that minimizes the total distance to other objects in the cluster; ;

    Replace m_c with the new medoid;

  end

  //Step 3: Assign objects to medoids

  Assign each object to the nearest medoid and obtain the clustering result;

  $d_{current}$=Calculate the sum of distance from all objects to their medoids;

until *convergence* ($d_{previous} = d_{current}$);

Return C;

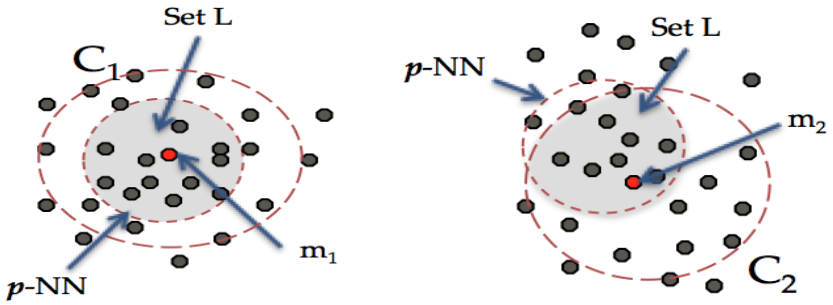
Additionally, the value of t can be set as the average of p-NN distances of all features across a sample of data objects, or can be set to any positive constant.

For each current cluster C_c, let L be a set of a medoid m_c, and its local neighbor objects defined by p-NN list and located in C_c (Figure 4.1 shows an illustration). Formally, L=$\{m_c \cup x_i \mid x_i \in C_c$, and $x_i \in m_c$'s p-NN list$\}$

We use a simple aggregation function, the average, to evaluate the discriminability of each feature per cluster. The average of wID score, $\text{wID}_{f-average}$, is computed along each feature f_j across the objects in L, as the following,

$$\text{wID}_{f-average} = \frac{1}{|L|} \sum_{x_i \in L} \text{wID}_i(f_j) \tag{4.4}$$

Equation (4.4) considers both the density of the local neighborhood around each object in the set L, and the complexity of local ID with respect to a particular feature f_j. The local neighborhood around each cluster representative in Equation (4.4) is used as a feedback to support finding the most discriminative features per cluster C_c (This will be further discussed in Section 4.2.2).



(a) Set L in cluster C_1 (b) Set L in cluster C_2

Figure 4.1 Examples of set L in two clusters, where L is defined as local neighborhood for each cluster representative, and p-NN is local neighbor objects list for each medoid.

4.2 k-LIDoids Algorithm

In this section, we give an overview of our proposed method, k-LIDoids. As stated before, the method interleaves the clustering process in terms of km-medoids (Section 4.1.2), with the feature selection criterion, support-weighted ID (wID) (Section 2.3.2), and with other original ideas in order to enhance the overall clustering quality. The algorithm starts with initial forms of clusters. It then iteratively and gradually eliminates (sparsifies) the least important (noisy) features, which have highest wID scores, for each cluster in order to find the intrinsic dimensions (subspace) for every cluster separately. In every iteration, the object is assigned to the cluster based on the current subspace defined for that cluster. This iterative process continues until the termination criterion is satisfied.

The complete pseudocode for our algorithm is given in Algorithm 4. The input parameters are k, Z, M, and p-NN graph G, where k is number of clusters, Z is a fixed proportion of features that are sparsified in each iteration per cluster, T is the total number of desired iterations, and p-NN graph G is the working neighborhood used for computing the wID values–in every iteration—for each object's features. The feature sparsification rate Z should be relatively small.

The algorithm works in two phases, an initialization phase, and an iterative phase as described below:

4.2.1 Initialization Phase

Using full features of X, the algorithm computes the initial best clusters C—using km-medoids clustering method presented in 4.1.2—after normalizing the original feature vectors of the dataset X. This is essential step since clusters with a reasonably high quality are needed for the subsequent iterative phase in order to define the correct subset of features, and reduce searching for the best set of medoids in each cluster. In this context, we introduce and define the subspace mask vectors MV as follows:

Definition 3 *(**Subspace Mask Vectors** MV)*
Let the C be the clusters of objects in the dataset X, and F is the full features space of X, then, MV is a set of vectors defined together with C such that the objects in a cluster $C_c \in C$ are closely clustered in the subspace defined in the vector $MV_c \in MV$. The subspace in MV_c will have much lower dimensionality than the full space F ($|MV_c| \ll |F|$).

Each MV_c is initialized to 1's (full features). When the feature f_j is selected later in the iterative phase and defined as a noisy (bad) feature for a cluster C_c, then, $MV_c[j]$ will be changed to the value of 0.

4.2.2 Iterative Phase

The goal of this phase is to find iteratively the best set of medoids and the important set of features (intrinsic dimensions) for each cluster, such that the objects are

Algorithm 4: k-LIDoids Subspace Clustering method

Input : Dataset X, the number of clusters k, sparsification rate Z, maximum number of iterations M, p-NN graph G

Output: Clusters C, Subset of Feature per cluster

//Initialization Phase
Normalize the original feature vectors of X;
Let D=the length of the feature vector of each $x_i \in X$;
Let F=$\{1, 2, \ldots, D\}$ be the set of features, where $f_j \in F$;
Run km-medoids clustering(X,k) to obtain initial clusters C;
Set the value of a to the average of p-NN distances computed over a sample of X objects.
Define subspace mask vectors MV, where MV_c associated with $C_c \in C$, $|MV|$=k, and $|MV_c|$ =D;
Initialize each MV_c to 1s;
//Iterative Phase
repeat
    foreach *cluster* $C_c \in C$ *(execute in parallel)* **do**
        //Step 1: Feature Ranking and Sparsification
        Define a rank vector RV_c, where is $|RV_c|$ =D;
        Let L=$\{m_c \cup x_i \mid x_i \in C_c$, and $x_i \in m_c$'s p-NN list$\}$;
        for *each object* $x_i \in L$ **do**
            Compute the wID score (Equation 4.3) of each feature f_j, where $MV_c[j]$=1;
        end
        Compute the average of wID scores ($\text{wID}_{f-average}$) for every f_j, across the objects in L (Equation 4.4) , and store $\text{wID}_{f-average}$ in $RV_c[j]$;
        Rank values in RV_c in descending order;
        Change the value of the top-ranked Z-proportion of MV_c to 0 according to RV_c;
        ;
        //Step 2: Update Medoids
        if $|C_c| \geq \alpha$ **then** Using MV_c, compute the mean μ of cluster C_c;
        Search for a new medoid among the closest β objects to μ, such that the new medoid is the object that minimizes the total subspace distance measure (Equation 4.5) to other objects in the cluster; ;
        else Using MV_c, search a new medoid among the objects in the cluster C_c , which is the object that minimizes the total subspace distance measure (Equation 4.5) to other objects in the cluster; ;
        Replace m_c with the new medoid;
        ;
        //Step 3: Refine Nearest Neighbors
        Using the subspace distance measure defined on MV_c (Equation 4.5):
        Update G by re-sorting m_c' p-NN and R-NNs lists;
        And for each object $x_i \in p$-NN list of m_c and $x_i \in C_c$:
        Update G by re-sorting x_i' p-NN and R-NNs lists;
    end
    //Step 4: Assign objects to medoids
    Using the subspace distance measure defined on MV (Equation 4.5):
    Assign each object to the nearest medoid m_c, and obtain the clustering result;
    ;
    Compute the overall average, $\text{wID}_{average}$, among the objects in the clusters C according to Equation 4.8
until *reaches the maximum* M*, or the minimum* $\text{wID}_{average}$ *(Equation 4.8)*;
Return (C, MV);

assigned to the cluster based on the subspace represented by this small subset of features. The algorithm consists of a number of iterations defined in the user input parameter M, in which each of the following four steps are executed.

Feature Ranking and Sparsification For each cluster C_c, the average of the wID scores, $\text{wID}_{f-average}$, is computed along each feature f_j—if its corresponding mask value,$MV_c[j]$, is set to 1— across the objects in the local neighborhood set, L, defined in Section 4.1.3.

To each cluster C_c, we also associate a rank vector RV_c that stores the values of $\text{wID}_{f-average}$ of all features. The values in RV_c are used to determine the best features locally per cluster, and also to exclude the noisy features.

For feature ranking per cluster C_c, a straightforward method is used for selecting the most local discriminative features stored in RV_c, in which the D features are ranked in descending order of $\text{wID}_{f-average}$, and a proportion Z of the top-ranked features are determined as candidates for sparsification as noisy features. The sparsification process will set the values of the least important features in MV_c to 0 (the feature global mean after normalization). The subspace that is defined in MV_c will help later in determining the objects that belong to the cluster C_c. The values in MV_c are gradually changed to 0's—depending on the feature sparsification value of Z in every iteration—which is important to avoid the loss of information from clusters; decreasing the full feature space sharply to a very lower-dimensional space may distorts the well forming of the clusters.

Update Medoids To update the current medoid of the cluster C_c, the algorithm searches for a new object that minimizes the total distance to other objects in C_c using the distance measure computed on the feature subspace defined in MV_c. For example, the Euclidean distance function d (defined in Equation 4.2) between a mediod m_c and an object x_i in a cluster C_c, using a subspace defined in MV_c, is computed as follows:

$$d(m_c, x_i) = \Big(\sum_{MV_c[j]=1, j=1}^{D} |m_{cf} - x_{if}|^2 \Big)^{\frac{1}{2}} \tag{4.5}$$

The search for a new medoid can be limited to the closet β objects (i.e., β=100) to the cluster mean, as described in Section 4.1.2, but with using the distance measure on the feature subspace defined for the cluster. This new medoid will replace the current medoid m_c.

Refine Nearest Neighbors For each current medoid m_c and its local neighborhood defined in the set L (as described in Section 4.1.3), the p-NN lists—defined in the whole dataset—are recomputed using the new subspace defined in MV_c. This is an important step in order to keep the consistency of the local neighborhood of the specified objects with the new subspace defined for cluster C_c, and thus, use correct neighbors for computing wID scores in the feature ranking step in the subsequent iteration. For efficiency, we limit the search for the new neighbors—in order to update the p-NN lists—among the R-NN objects. In the implementation, we set p to be the length for both R-NN and NN lists used in the refine nearest neighbors step

Assign Objects to Medoids Given the current updated set of medoids, the algorithm assigns each object to the closet medoid based on the clustering subspaces defined in MV. In other words, this step computes the distance value (using Equation 4.5) between each object x_i and meodid m_c using the feature subset defined in MV_c, and the object is assigned to a cluster C_c which this value is the least. In a given iteration, this step serves to refine the objects in the clusters.

4.2.3 Termination Criteria for the Clustering Convergence

The algorithms shall be terminated when either of the following defined criteria is occurred: First, the maximum number of iterations defined in the parameter M is satisfied. Second, after reaching the minimum value, the average wID ($\text{wID}_{average}$), which is computed among all clusters, increases. In this context, we explain the second heuristic criterion in more detail.

Equation 2.10 can be used for finding the support-weighted ID score per object x_i, with respect to all features, as the following:

$$\text{wID}(x_i) = \text{ID}_i \cdot \left(\frac{t}{w_i}\right)^{\text{ID}_i} \tag{4.6}$$

where ID_i here is defined as the local intrinsic dimensional estimate for the neighborhood around an object x_i, and w_i is the distance to the p-th nearest neighbor of the object x_i defined in the graph G. As we stated before, the value of t can be set to any positive constant.

Using the local neighborhood objects of each medoid m_c defined in the set L (Section 4.1.3), the average wID score for a particular cluster C_c is computed as the following,

$$\text{wID}_{c-average} = \frac{1}{|L|} \sum_{x_i \in L} \text{wID}(x_i) \tag{4.7}$$

If $\text{wID}_{c-average}$ is high, then the relative number of objects in the neighborhood of the cluster representative m_c is expected to be large. Thus, the density of the cluster C_c is high, and the objects in C_c have a high dimensionality. On contrary, if $\text{wID}_{c-average}$ is low, then the objects in C_c have a low dimensionality. Removing excessively many features—including the most discriminative features—from a cluster C_c will increase the cluster density; the cluster may contains additional random objects from other clusters. Therefore, the $\text{wID}_{c-average}$ value of the cluster C_c will become high.

The overall average weighted ID across all clustersis defined as,

$$\text{wID}_{average} = \frac{1}{|k|} \sum_{c=1}^{k} \text{wID}_{c-average} \tag{4.8}$$

The algorithm computes the $\text{wID}_{average}$ score in every iteration. When the minimum value of this score is found, the value of the next iteration is checked. If the $\text{wID}_{average}$ score starts to increase, then, it is expected that the overall clustering quality starts to drop, and the loop terminates. In general, Equation 4.8 helps to assist the subset

of features preserved so far of all clusters and the overall clustering quality. Increasing $\text{wID}_{average}$ after reaching the minimum score means that the algorithm starts to prune important features from the clusters, and thus, the cluster density increases by including irrelevant objects from different other clusters.

4.2.4 Time Complexity

The time complexity of k-LIDoids can be divided according to its phases as follows: For the initialization phase, data normalization and km-medoids clustering —in terms of distance computation until convergence— take $O(Dn)$ and $O(n^2D)$, respectively. Computing the values of t using the average of p-NN distances takes $O(Dpn)$. For each iteration of the iterative phase, feature ranking and sparsification using wID takes $O(pD + D\log D)$ per medoid, with a total time is $O(p^2Dk + pkD\log D)$ taken over all medoids and their neighbors. Updating medoids step takes $O(n^2D)$, but the simple indexing technique defined in Section 4.2.2 can also be applied here to speed up this step. Since we are assuming that the lengths of the RNN lists are in $O(p)$, each iteration of k-LIDoids takes $O(p^3Dk)$ for the refine nearest neighbors step. For improving the scalability of the algorithm, steps 1-3 of every iteration can be executed in parallel for all clusters. Finally, the step of assigning objects to the medoids takes $O(kn)$.

4.3 Experimental Framework

4.3.1 Competing Methods

The performance of k-LIDoids is compared and contrasted with four competitors:

- **Random**: as per k-LIDoids, except that for each object, the features to be sparsified are selected randomly. The rationale for the comparison with this method is to establish a baseline for the performance of the feature ranking and sparsification criterion used in k-LIDoids.

Among several subspace clustering algorithms in the literature, we select the most well-known clustering methods, PROCLUS and CLIQUE. We exclude other

algorithms such as MAFIA, FINDIT and δ-cluster. While MAFIA is just a successor of CLIQUE with more efficiency, FINDIT and δ-cluster require more information about the datasets for accurately determining their parameters setting.

- **PROCLUS**: is the most similar well-known algorithm to k-LIDoids. As described in Section 2.1.2, PROCLUS uses k-medoids algorithm, specifically CLARANS [142], as the basic framework for the defined subspace clustering.

- **CLIQUE**: is a grid-based subspace clustering algorithm. As described in Section 2.1.2, CLIQUE divides each dataset dimensions into multiple cells, then combines adjacent high-density cells of all dimensions to form the final clustering result.

From different global feature evaluation techniques that are combined with clustering methods, we select the simple correlation model that combines PCA with the k-means algorithm, as the other algorithms highly depend on the accuracy of the parameters tuning, which is in general difficult to define.

- **Correlation Model (PCA and k-means clustering)**: In this method, the feature extraction and k-means clustering are conducted as two separate processes. To allow a fair comparison with other methods, after choosing the highest principal components, the k-means clustering algorithm is executed.

4.3.2 Datasets

Six real datasets of varying sizes and densities were considered:

- **ALOI-100** is subset of the Amsterdam Library of Object Images (ALOI) [135], which contains 110,250 images of 1000 small objects. Each image is described by a 641-dimensional feature vector based on color and texture histograms. ALOI-100 contains 10,800 images of 100 simple objects generated by selecting the objects uniformly from among the classes.

- **Wearable Computing** Classification of Body Postures and Movements (PUC-Rio) dataset [145] contains 165,633 samples collected on eight hours of activities of four healthy subjects in different static postures and dynamic movements. Each sample's features vector has 18 attributes that represent user data such as name, gender, age, height, weight, body mass, and sensor axis values. There are five possible positions (sitting-down, standing-up, standing, walking, and sitting).

- **Online News Popularity dataset (ONP)** [146] contains 39,644 articles' textual extracted data, each has 60 attributes (58 predictive attributes, and 2

non-predictive) that describe different article aspects. The articles are binary classified as popular and unpopular using a decision threshold of 1400 social interactions.

- **MAGIC** Gamma Telescope dataset [147] is generated to simulate registration of high energy gamma particles in an atmospheric Cherenkov telescope. It contains 19,020 cases of 10 numerical predictors (attributes) and 2 classes. The predictors are produced by the registration device and characterize the registered particle.

- **MiniBooNE** [138] particle identification dataset contains 130,065 of signal and background events Each event has 50 particle ID variables. This dataset is taken from the MiniBooNE experiment and is used to distinguish between two classes, electron neutrinos (signal), and muon neutrinos (background).

- **Sensorless Drive Diagnosis dataset (SDD)** [138] includes 58,509 electric drives, with 48 features extracted form electric current drive signals. The drive has intact and defective components. This results in 11 different classes with different conditions.

4.3.3 Parameters Setting

In k-LIDoids, implemented in JAVA, for all datasets, the value of t in the weight parameter of Equations (4.3 and 4.6) is set to be the average of p-NN distances over a random sample of 100 objects. Furthermore, for simplicity, the value t is precomputed in advance using the original feature vectors in the initialization phase of the algorithm. The choices of Z are varied with different datasets as it heavily depends on the density of the feature vectors. The number of clusters k is fixed to be equal to the ground truth labels for the datasets. The value of p for the p-NN graph is set to 30 for the relatively small datasets ($n \leq 50,000$) such as MAGIC and ONP, and 100 for the other large datasets. The threshold for the cluster size, α, is set to $100,000$ objects, and the threshold for the number of objects that are close to the cluster mean, β, is set to 100 objects. Table 4.1 summarizes the datasets information and the parameters setting for the Z value and the maximum number of iterations M.

In PROCLUS, executed using ELKI source-code [148], the average number of dimensions to be kept in each cluster per execution is set to be equal to (D-current

iteration$*D*Z$), and the multiplier for the initial number of seeds and medoids are set to, $\sqrt{n}$ and 100, respectively.

In the correlation model (PCA and k-means clustering), executed using the scikit-learn Python library [149], the number of principle components per execution is set to (D-current iteration$*D*Z$) as well, and the maximum number of iterations used for k-means is fixed to 1000 or until convergence.

In CLIQUE, also executed using ELKI source-code [148], we found that every dataset in comparison requires a strict tuning for the pair of parameters, the density threshold (ϵ) and the grid size ($gridSize$). Table 4.2 shows the different parameters setting that we have chosen for every dataset. We fixed the density threshold of each dataset, and increased the grid size by a constant in every iteration. We tried many other parameter settings as well, but the algorithm crashed due to insufficient memory problem (memory size=16GB). We did not include ALOI-100 and MiniBooNE in Table 4.2 as CLIQUE always crashes for these datasets as they have a large number of objects and dimensions combined.

In order to smooth the curves of each experiment, the results were averaged over 40-140 runs, depending on the result variations in each dataset.

4.3.4 Evaluation

For each of the considered datasets, the clustering results of the proposed method are compared with the other competing algorithms, and assessed against the ground truth classification labels using two different quality measures: Adjusted Rand index (ARI) [150] from statistics; and the Expected Precision (EPrec), Expected Recall (ERec), and Expected Cosine (ECos) measures [151] from information retrieval.

Let C denotes the set of clusters that are partitioning the data objects according to the clustering algorithm, and $\mathcal{O}$ denotes the classification of the dataset, then we define each measure as follows,

Table 4.1 Datasets used in the Experiments

Datasets	*Instances (n)*	*Features (D)*	*#True Classes*	*Sparsification rate (Z)*	*Number of iterations (M)*
ALOI-100	10,800	641	100	0.0025%	40
Wearable Computing	165,633	18	5	0.055%	15
ONP	39,644	60	2	0.02%	29
MAGIC	19,020	10	2	0.1%	7
MiniBooNE	130,065	50	2	0.04%	24
SDD	58,509	48	11	0.04%	23

Table 4.2 CLIQUE Parameters Setting for the Density Threshold (ϵ), and the Grid Size ($gridSize$)

Datasets	*density threshold (ϵ)*	*gridSize*	*The increase in gridSize in every iteration*
Wearable Computing	0.2	10	+1
ONP	0.7	10	+10
MAGIC	0.5	2	+2
SDD	0.2	100	+10

Adjusted Rand index (ARI): ARI is used to measure the global quality of clustering results, and is calculated as:

$$ARI = \frac{2 * (ad - bc)}{(a + b)(b + d) + (a + c)(c + d)} \tag{4.9}$$

where a is the number of pair of objects that belong to the same true class in $\mathcal{O}$ and placed in the same cluster in C , b is the number of pairs that belong to the same true class in $\mathcal{O}$ but placed in different clusters in C, c is the number of pairs that placed in the same cluster in C but they belong to different true classes in $\mathcal{O}$, and d is the number of pairs that belong to different true classes in $\mathcal{O}$ and placed in different clusters in C.

Expected Measurements: Let O defines a unique class, where $O \in \mathcal{O}$, and $\mathcal{C}$ defines a unique cluster, where $\mathcal{C} \in C$. Then, expected precision (EPrec), expected recall (ERec), and expected cosine (ECos) are given by:

$$EPrec = \frac{1}{n} \sum_{\mathcal{C} \in C} \sum_{O \in \mathcal{O}} \frac{|\mathcal{C} \cap O|^2}{|\mathcal{C}|} \tag{4.10}$$

$$ERec = \frac{1}{n} \sum_{\mathcal{C} \in C} \sum_{O \in \mathcal{O}} \frac{|\mathcal{C} \cap O|^2}{|O|} \tag{4.11}$$

$$ECos = \frac{1}{n} \sum_{\mathcal{C} \in C} \sum_{O \in \mathcal{O}} \frac{|\mathcal{C} \cap O|^2}{\sqrt{|O| \cdot |\mathcal{C}|}} \tag{4.12}$$

For both, ARI and expected measurements, the highest possible value is 1, which indicates a better clustering, and a perfect agreement between classification labels and clustering results. A low expected precision score means few clusters are generated and thus cluster fusion. On the other hand, a low expected recall score means many clusters are produced and thus cluster fragmentation. Finally, achieving a high expected cosine score means that the clustering method is avoiding extremes of cluster fusion and cluster fragmentation, and that the number and size of the clusters are almost follow those of the ground truth classification labels.

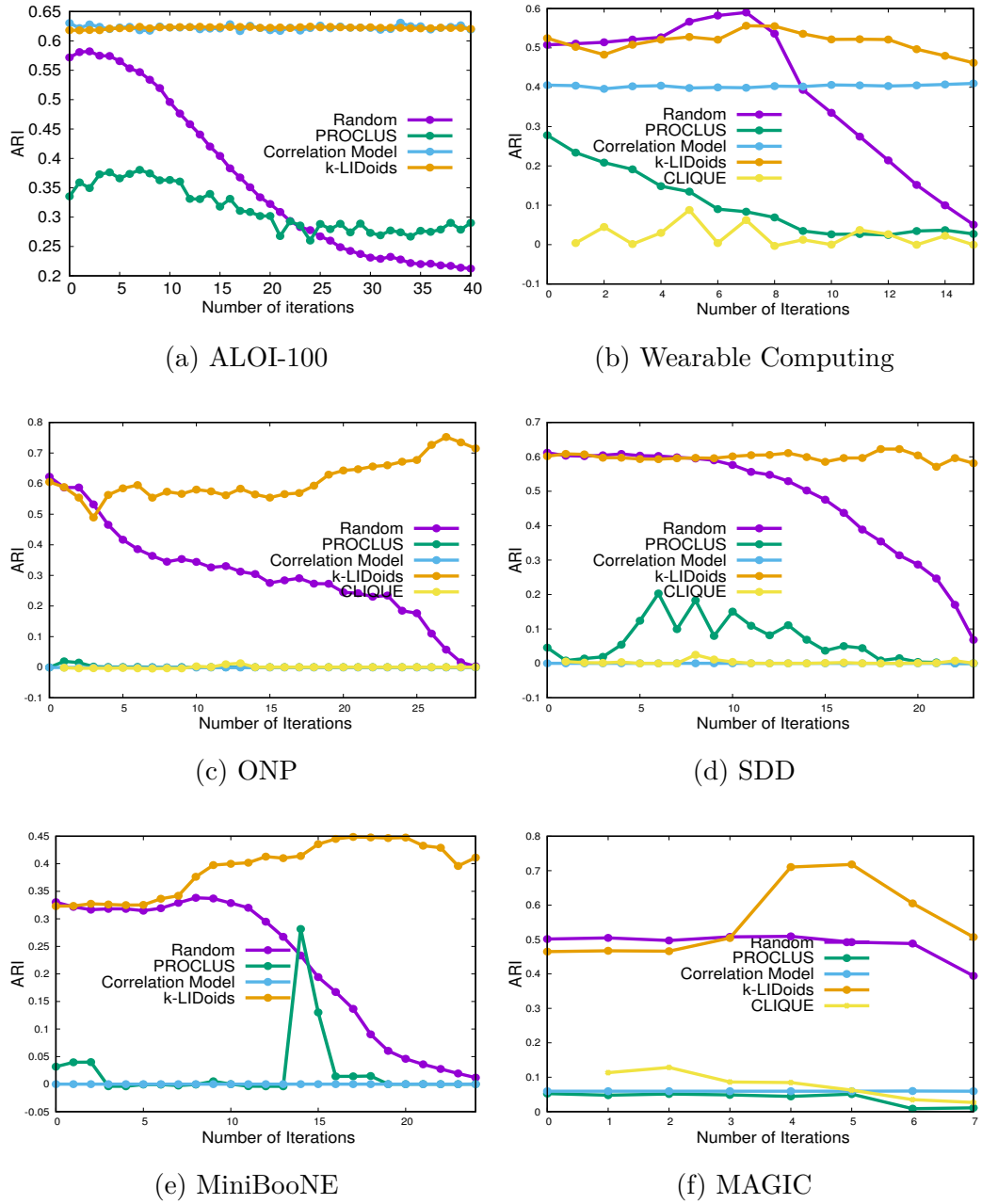
4.3.5 Comparison Against the Competing Methods with Respect to ARI and the Maximum Number of Iterations M

In this experiment, we compared between the proposed method and other competitors in terms of ARI. The second termination criterion ($\text{wID}_{average}$) has been disabled in order to test the resistance of the proposed algorithm toward pruning many features away from the clusters. Figure 4.2 shows plots of the ARI results of clustering for all the methods, across a range of iterations over the Z values of each dataset presented early in Table 4.1.

Results and Analysis Over all datasets used, in compared with its competitors, k-LIDoids achieves consistent improvements for clustering accuracy and resistance to the performance degradation as the sparsification increases. For ALOI-100, the Correlation Model has a performance comparable to k-LIDoids. However, on four of the six datasets, we notice that the Correlation Model has a very low performance. It worth mentioning that in the iteration 0, the actual k-means clustering–without any feature reduction method–is applied on the datasets. Therefore, if the initial resulted clusters are poorly performed, applying any dimensionality reduction technique such as PCA will not show much improvement in the overall clustering performance. In addition, the defined Correlation Model uses k-means clustering which itself sensitive to the outliers.

In general, PROCLUS has an unstable behavior with changing the number of the average number of features to be kept in the clusters. For example, in MAGIC and ONP, PRCOLUS achieves a low performance even with retaining a large number of dimensions in the clusters.

We noticed that the performance of CLIQUE algorithm is the worst in terms of the scalability with respect to the number of objects and features. This is a common problem of grid-based clustering algorithms with high-dimensional datasets; when



(a) ALOI-100 (b) Wearable Computing

(c) ONP (d) SDD

(e) MiniBooNE (f) MAGIC

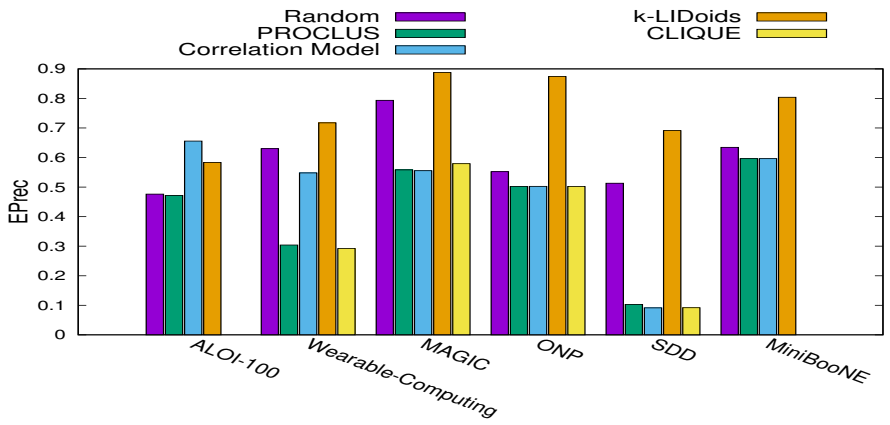
Figure 4.2 Adjusted Rand Indices (ARI) for all methods in comparison with respect to the total value of sparsification rate Z as, 12% for ALOI-100 (Total Sparsified Features =80/641), 83% for Wearable Computing (Total Sparsified Features =15/18), 97% for ONP (Total Sparsified Features =58/60), 70% for MAGIC (Total Sparsified Features =7/10), 96% for MiniBooNE (Total Sparsified Features =48/50), and 96% for SDD (Total Sparsified Features =46/48).

the number of dimensions increases, the number of cells grows exponentially and thus finding clusters in adjacent high-density cells becomes prohibitively expensive [152].

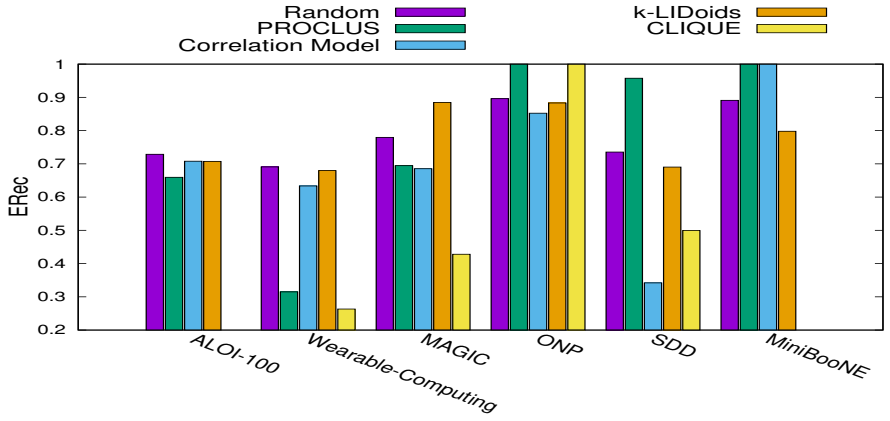
Therefore, we can conclude that PROCLUS, the Correlation Model, and CLIQUE clustering accuracies considerably depend on both the parameters setting— which is, however, difficult to determine— and the actual datasets used in the experimentation.

4.3.6 Comparison Against the Competing Methods with Respect to the Expected Measurements

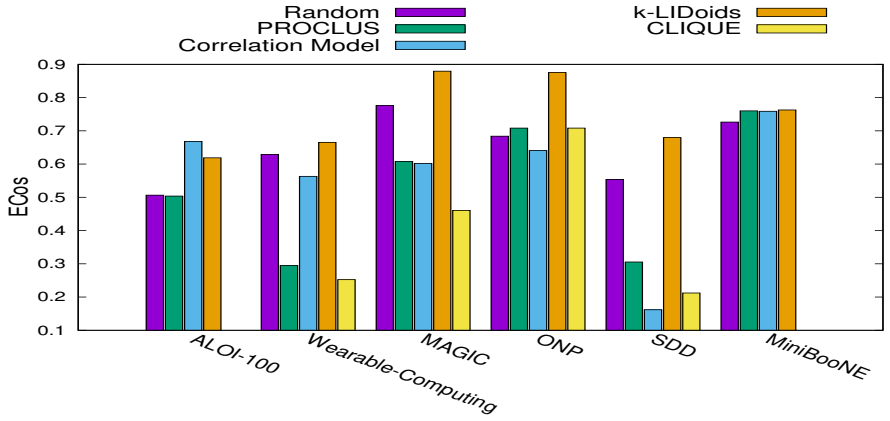
Figure 4.3 shows the plots of EPrec, ERec, and ECos for a specific iteration of each dataset. Specifically, we use iterations 30 for ALOI-100, 9 for Wearable Computing, 5 for MAGIC, 27 ONP, 18 for SDD, and 19 for MiniBooNE; these iterations represent the highest ARI values in Figure 4.2



(a) Expected Precision



(b) Expected Recall



(c) Expected Cosine

Figure 4.3 Expected Precision, Recall, and Cosine values for all methods in comparison in a specific iteration (M, Z) as (30, 0.093) for ALOI-100, (9, 0.5) for Wearable Computing, (5,0.71) for MAGIC, (27,0.9) for ONP, (18,0.75) for SDD, and (19,0.76) for MiniBooNE.

Results and Analysis Except for ALOI-100, k-LIDoids outperforms other methods in terms of EPrec and ECos scores, which means that our clustering method, in most cases, avoids extremes in cluster fusion and cluster fragmentation. When compared with the other competing methods, even though k-LIDoids has the lowest value for the ERec measurement, nevertheless, its performance is considerably acceptable in most datasets (i.e., $\geq 70\%$).

4.3.7 Comparison Against the Competing Methods with Respect to the Clustering Convergence

In this experiment, we enabled both termination criteria for the k-LIDoids algorithm, the maximum number of iterations M and the overall average weighted ID ($\text{wID}_{average}$). Except for ALOI-100, we used the same setting defined in Table 4.1 for the parameter M for all datasets in comparison. For ALOI-100, we set M to 200 iterations, as to test if we can eliminate more noisy features from this dataset without reducing the clustering performance. Figure 4.4 shows the plot of the comparison results between k-LIDoids, PROCLUS, and the Correlation Model with respect to ARI after all methods converged. We did not include CLIQUE as it has a different execution style. For a fair comparison, both parameters, the average number of dimensions in PROCLUS and the number of the principal components in the Correlation Model are explicitly set to 400 for ALOI-100, 9 for Wearable Computing, 5 for MAGIC, 48 for ONP, 20 for SDD, and 38 for MiniBooNE; these values reflect the number of dimensions that is kept in each dataset after executing the k-LIDoids algorithm.

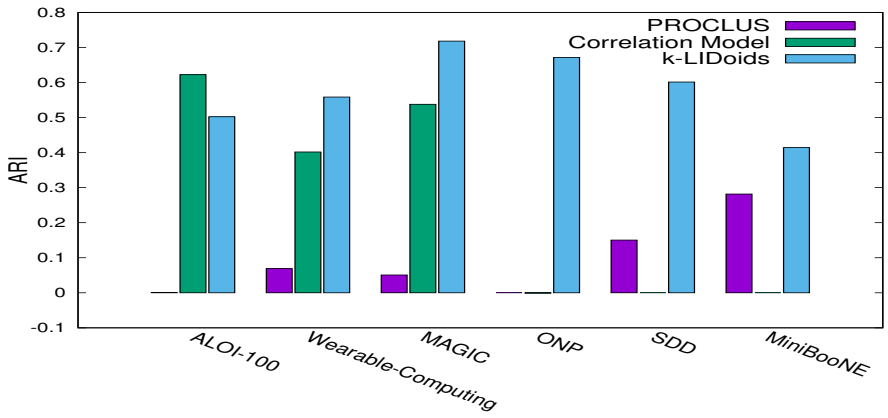


Figure 4.4 ARI values for k-LIDoids, PROCLUS, and the Correlation Model with respect to the clustering convergence. k-LIDoids reaches a specific iteration and sparsification rate before terminated (M, Z, $\text{wID}_{average}$), as (200, 0.62, N/A) for ALOI-100, (9, 0.5, 3.89E+36) for Wearable Computing, (5,0.71, 0.310386162) for MAGIC, (24,0.8, 0.223351866) for ONP, (10,0.42, 3.09E+36) for SDD, and (19,0.76, 0.022918872) for MiniBooNE.

Results and Analysis Except for ALOI-100, k-LIDoids is terminated before reaching the maximum number of iterations, and outperforms other methods in all datasets, see Figure 4.4. The Correlation Model, however, achieved the highest ARI value in ALOI-100 dataset. In four out of the six datasets used in the experiments (i.e., MAGIC, Wearable Computing, ONP, and MiniBooNE), we observe that the minimum value for the termination criterion $\text{wID}_{average}$ occurs when k-LIDoids achieves relatively the highest performance in Figure 4.2. However, in SDD dataset, the k-LIDoids algorithm is terminated before it reaches its highest performance (where M=18 in Figure 4.2). In ALOI-100, however, k-LIDoids is terminated when it reaches its maximum number of iterations (M=200). In fact, applying k-LIDoids on ALOI-100 dataset does not show a clear variations for the $\text{wID}_{average}$ value in every iteration. The possible explanation for this behavior is that ALOI-100 consists of too many clusters, with a relatively small number of objects per cluster, that prevents significant clustering changes to occurs in each iteration. In general, these

observations still show some usefulness of using the defined heuristic termination criterion $\text{wID}_{average}$.

4.3.8 Case Studies for the Clustering Accuracy

In this section, we present the confusion matrices of the outcomes of k-LIDoids applied on the datasets with a small number of ground truth labels. The goal here is to test the accuracy of the proposed algorithm while it progresses in gradually removing the noisy features as well as clustering the data objects in the subspaces. In each confusion matrix, each entry value is equal to the number of objects belong to the true class O, and assigned to a cluster $\mathcal{C}$. Clearly, each row should have one entry that is much lager than the other entries.

We show the confusion matrix for the initial clustering results without any sparsification, and also the confusion matrices after sparsification for each iteration that has ARI score among the largest ARI scores shown in Figure 4.2. Tables 4.3 through 4.6 show the results of four datasets with a small number of ground truth labels, namely, MAGIC, MiniBooNE, and ONP, and Wearable Computing. Obviously, the results are for one execution of multiple runs of the k-LIDoids algorithm. The iteration numbers are chosen to reflect the significant changes in the clustering process, as the clustering performance is not reflected clearly in the initial iterations.

As can be seen from the tables, every class objects are directed to the correct cluster with the exception for some objects. For three datasets with a binary classification, the percentage of these misplaced objects is relatively small which does not affect the correspondence between the true classes and the resulted clusters. This indicates a clean mapping form the ground truth labels to the generated clusters with increasing the number of iterations, and with sparsifying more features in order to define cluster subspaces. For Wearable Computing dataset, the number of misplaced

Table 4.3 k-LIDoids: Confusion Matrix for MAGIC

Iteration:0, ARI=0.3749		
Cluster Class	1	2
1	8649	0
2	3683	6688

Iteration:3, ARI=0.4982		
Cluster Class	1	2
1	9535	0
2	2798	6687

Iteration:5, ARI=0.9939		
Cluster Class	1	2
1	12303	0
2	29	6688

Table 4.4 k-LIDoids: Confusion Matrix for ONP

Iteration:0, ARI=0.4604		
Cluster Class	1	2
1	12124	0
2	6366	21154

Iteration:16, ARI=0.7853		
Cluster Class	1	2
1	16234	0
2	2256	21154

Iteration:29, ARI=0.8303		
Cluster Class	1	2
1	18490	1760
2	0	19394

Table 4.5 k-LIDoids: Confusion Matrix for MiniBooNE

Iteration:0, ARI=0.59626		
Cluster Class	1	2
1	36499	14667
2	0	78898

Iteration:18, ARI=0.92614		
Cluster Class	1	2
1	36499	2370
2	0	91195

Iteration:24, ARI=0.891172		
Cluster Class	1	2
1	36499	3534
2	0	90031

Table 4.6 k-LIDoids: Confusion Matrix for Wearable Computing

Iteration:0, ARI=0.32688					
Cluster Class	1	2	3	4	5
1	19801	0	0	0	0
2	30830	11827	14575	0	0
3	0	0	32795	12415	10340
4	0	0	0	0	18952
5	0	0	0	0	14098

Iteration:9, ARI=0.53408					
Cluster Class	1	2	3	4	5
1	22,722	0	0	0	0
2	27909	1987	0	0	0
3	0	9840	41657	0	0
4	0	0	5713	12415	35141
5	0	0	0	0	8249

objects is still high, but as can be seen from Table 4.6, more and more objects are moved to the correct clusters.

Another interesting result is the subset of features defined by k-LIDoids for the resulted clusters in each iteration. Tables 4.7 through 4.10 illustrate the subset of features that is kept per cluster for each dataset, particularly, in iterations 3 and 4 for MAGIC, 29 for ONP, 24 for MiniBooNE, and 9 for Wearable Computing.

Table 4.7 k-LIDoids: Subset of Features per Cluster in MAGIC

Iteration:3, ARI=0.49825	
Cluster 1	2,3,4,5,7,8,9
Cluster 2	4,5,6,7,8,9,10
Iteration:5, ARI=0.99386	
Cluster 1	2,3,4,5,8
Cluster 2	5,7,8,9,10

Table 4.8 k-LIDoids: Subset of Features per Cluster in ONP

Iteration:29, ARI=0.8302986	
Cluster 1	51, 54
Cluster 2	2, 51

Table 4.9 k-LIDoids: Subset of Features per Cluster in MiniBooNE

Iteration:24, ARI=0.891172	
Cluster 1	27, 34
Cluster 2	34, 41

From the results of the feature subsets–even with the lack of information about the actual numerical features that distinguish each ground truth class–and

Table 4.10 k-LIDoids: Subset of Features per Cluster in Wearable Computing

Iteration:9, ARI=0.53408	
Cluster 1	7,8,9,10,11,12,13,16,17
Cluster 2	7,8,9,12,13,15,16,17,18
Cluster 3	7,8,9,13,14,15,16,17,18
Cluster 4	7,8,9,11,12,13,16,17,18
Cluster 5	7,8,9,10,11,12,13,16,17

the clustering accuracy values, we still can conclude the intrinsic dimensions that represent each cluster. This is crucial for some applications that require not only good clustering results for the dataset but also additional information regarding to the minimum subset of features representing each cluster.

4.4 Conclusion

In this chapter, we proposed k-LIDoids as a subspace clustering algorithm for discovering similar objects in a subset of features (subspace) for high-dimensional datasets. k-LIDoids integrates k-medoids clustering algorithm with the feature selection criterion, Support-Weighted Intrinsic Dimensionality (support-weighted ID). Support-weighted ID is used to identify the relevant features with a higher discriminative power per cluster in order to define each cluster subspace.

Experimental results have shown that k-LIDoids is able to return the clusters of data objects together with the subset of features defining each cluster with maintaining or increasing the overall clustering accuracy in compared to clustering the full-dimensional dataset. The performance of k-LIDoids is also more stable and resistance toward excessively eliminating many noisy features from the clusters in compared with other state-of-the-art algorithms such as PROCLUS and CLIQUE.

CHAPTER 5

LID-FINGERPRINT: A LOCAL INTRINSIC DIMENSIONALITY-BASED FINGERPRINTING AND INDEXING METHOD FOR SIMILARITY SEARCH

In this chapter, we define LID-Fingerprint, a fingerprinting and multi-level indexing framework that can be used for hiding the information on a server side (the cloud) as a way of preventing passive adversaries. LID-Fingerprint combines the k-LIDoids algorithm, presented in Chapter 4, as a subspace clustering that can define a minimum subspace for each cluster, and NNWID-Descent [153], presented in Chapter 3, as a similarity graph construction method based on support-weighted ID, which can be used to obtain sparse representations for dataset objects. In LID-Fingerprint, k-LIDoids is used in a hierarchal nesting of a subspace clustering framework that allows to define a common binary fingerprint for each group of objects in each intermediate level of the index. On the other hand, the leaf level holds the fingerprints that derived from the sparse representations of the objects, which are resulted from using NNWID-Descent.

By combining dimensionality reduction (i.e., feature selection) and fingerprints generation, our method provides two of the standard measures for protecting the data privacy: *data suppression* and *data masking*. Furthermore, since our method does not assume the uniqueness of the generated fingerprints (several objects of the same neighborhood can have similar fingerprints), therefore, it does provide data anonymity. With using an appropriate similarity measure, the generated indexed fingerprints allow to achieve a reasonable similarity search accuracy. The main contributions are:

– The proposed method allows an efficient and secure search and retrieval– as only fingerprints represented as binary vectors are used–without revealing any information about the actual values for the objects features.

– The search results of the related fingerprints to a given query are considered a proper superset of the actual results, which then can be refined, in the client side, using the actual object representations in order to achieve a smaller more accurate results.

– Using real datasets, the proposed method is compared against other state-of-the-art methods such as Locality Sensitive Hashing based approaches.

The remainder of this chapter is organized as follows. We discuss the proposed LID-Fingerprint in Section 5.1. In Section 5.2, the performance of our method, along with the experimental results and analysis using several real datasets, is compared to NNWID-Descent and other competing methods from the literature. Finally, we conclude the discussion in Section 5.3.

5.1 LID-Fingerprint Framework

In this section, we present LID-Fingerprint, a new fingerprinting and indexing framework, that includes three fundamental processes: fingerprinting, indexing, and nearest neighbor search. The fingerprinting process defines a set of binary fingerprint vector representations for the objects represented as vectors in high-dimensional spaces. These fingerprints can be used in a filtering phase to select candidates for similarity search based on the Hamming distance. In large datasets, the number of these fingerprints can be high and the similarity can be extensive to compute. Thus, the indexing process builds a multi-level (hierarchical nesting of subspace clustering) data structure (index) that allows reducing the number of computations, and speeding up the search for these fingerprints. Given a fingerprint for a query object, the nearest neighbor search process searches the index for the best matched fingerprints based on a defined similarity measure (i.e., Hamming distance).

Similar to SUSHI [60] and $\triangle^+$-tree [63], LID-Fingerprint allows multiple representations for the objects according to different dimensions. Nevertheless, LID-Fingerprint uses binary vector representations (fingerprints) instead of the actual

values for the dimensions. This framework not only reduces the search space, but also allows a better usage for the main memory.

5.1.1 Notations

Let $X = \{x_1, x_2, x_3, \ldots, x_n\}$ be a dataset consisting of n objects such that each object x_i is represented as a feature vector in $\mathbb{R}^D$. Let the set of features is denoted as F=$\{1, 2, \ldots, D\}$ such that $j \in F$ is the j-th feature in the vector representation and D is the total number of features. We also define p as a neighborhood size for each object $x_i \in X$ in the p-nearest neighbor graph G. Suppose that these n objects should be partitioned, according to some distance measurement, into $k(k < n)$ clusters, which is assumed to be given. The set of clusters is denoted as C, where each cluster C_c, $1 \leq c \leq k$, has a cluster representative m_c.

5.1.2 Fingerprinting Process

For a better explanation of LID-Fingerprint framework, we first provide the intuition behind using the NNWID-Descent framework (presented in Chapter 3) in the objects fingerprinting. Then, we show the adoption of k-LIDoids subspace clustering (presented in Chapter 4) in generating cluster fingerprints.

Objects Fingerprinting Given a dataset X, we rank and select the best features locally for each object $x_i \in X$ using weighted ID (Equation 3.3, Chapter 3) on the normalized feature vectors, and sparsify each vector to remove the noisy features (Section 3.2, Chapter 3). This process will project each object x_i onto a subset of features $F' \subset F$ (the set of features) to provide a compact representation, x'_i, where for all $j \in F$, feature x'_{ij}=x_{ij}, whenever $j \in F'$, and $x'_{ij} = 0$, otherwise.

The compact representation x'_i of each object $x_i \in X$ can be further transformed into a binary vector x''_i of $\{0, 1\}^D$ bits, such that for all $j \in F$, feature $x''_{ij} = 1$, whenever $j \in F'$, and $x'_{ij} = 0$, otherwise. We call x''_i the LID-based fingerprint of x_i.

Objects Fingerprinting Algorithm Algorithm 5 illustrates the fingerprinting process. The only change made to NNWID-Descent (Algorithm 2, Chapter 3) is by modifying and adding few lines for defining the object fingerprints. More precisely, the binary fingerprints for all objects are initialized to 1's (line 2). Once $Z * T$ proportion of the object features are sparsified, their corresponding fingerprint bits are set to 0's (lines 18-21). The algorithm returns the set of binary fingerprints X'' instead of the p-NN graph G (line 22).

Algorithm 5: Objects Fingerprinting Process (only the modified and the new steps are shown - See Algorithm 2 in Chapter 3)

Input : Dataset X, distance function dist, neighborhood size p for the graph, neighborhood size K for computing wID scores, sparsification rate Z, number of iterations T

Output: Binary Fingerprints X''

2 Normalize the original feature vectors of X; Initialize X'' fingerprints to 1's;

18 **foreach** *data point* $x \in X$ **do**

19 | For each feature f in x, **if** $f \neq 0$ **then** set X''_{x_f} to 1 ;

20 | **else** set X''_{x_f} to 0;

21 **end**

22 Return X''

Clusters Fingerprinting For each cluster C_c, k-LIDoids creates what we refer to as *subspace mask vector* or *Cluster Fingerprint*, CF, as the following (Figure 5.1) ,

Definition 4 *(**Cluster Fingerprint** CF)*
Let the C be the clusters of objects in the dataset X, and F is the full feature space of X, then, CF is a set of binary fingerprint vectors defined together with C such that the objects in a cluster $C_c \in C$ are closely clustered and compact in a subspace defined in a fingerprint vector $CF_c \in CF$. The subspace in CF_c will have much lower dimensionality than the full space F ($|CF_c| \ll |F|$).

Each CF_c is initialized to 1's (full features). When the feature f_j is selected and defined as a noisy (bad) feature for a cluster C_c, then, $CF_c[j]$ will be changed to the value 0.

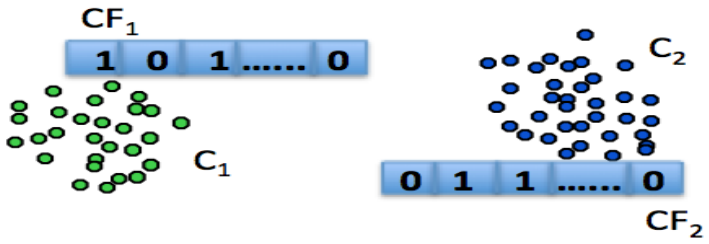


Figure 5.1 Binary fingerprint vectors, CF_1 and CF_2 defined together with clusters C_1 and C_2, respectively.

k-LIDoids clusters the objects using the subspace dimensions defined in CF such that objects are highly correlated and connected within those dimensions.

Clusters Subspace Based on Fingerprinting In this research, we allow k-LIDoids to generate a set of outliers beside the subspace clusters in order to increase the clustering quality and thus the overall indexing performance. Therefore, the subspace clustering generated by k-LIDoids can be defined, comparable to [60], as follows,

Definition 5 ***(Subspace Clustering based on k-LIDoids)***
Let CF be a set of the cluster fingerprints defined by k-LIDoids for clusters C of X. A subspace cluster C_c is a group of objects that are relevant according to the dimensions, defined in CF_c. Then, the subspace clustering $subspace\_C$ is defined as, $subspace\_C = (C_1, ..., C_c, .., C_k, Outliers\_set)$ where $1 \leq c \leq k$, and $Outliers\_set$ is the list of objects that are not relevant to any cluster C_c, $Outliers\_set = X - \bigcup_{c=1}^{k} C_c$

5.1.3 Indexing Process

In order to generate object fingerprints and synchronously create the multi-level index structure (tree), we combine the k-LIDoids algorithm as a method of creating cluster

fingerprints CF, with the objects fingerprinting process (Algorithm 5) which allows us to generate binary fingerprints for dataset objects. In each inner level of the index, each inner node stores cluster fingerprints generated by recursively applying k-LIDoids subspace clustering on a subset of the dataset. The Leaf node, on the other hand, stores the fingerprints of the objects derived from the sparse representations of NNWID-Descent results. As determined by Definition 5, we obtain from the k-LIDoids algorithm both multiple clusters and an outliers set. The outliers fingerprints set is also derived from applying the objects fingerprinting process on the outliers, and stored in the inner node with its respective sub-clusters.

In this context, we define the nodes in LID-Fingerprint index structure as follows:

- **Inner Node IN and the root**: is determined by the cluster fingerprints CF and the outliers fingerprints set $Outliers\_set_{x''}$ as the following: IN= (CF_1, CF_2, ..., CF_c, ... , CF_k, and $Outliers\_set_{x''}$).
  - For each CF_c, the node contains a reference (or pointer) to its child node, which will represent the subset of objects represented by CF_c.
- **Leaf Node LN**: is represented by the cluster object fingerprints set, $C_{x''}$, where $C_{x''}$=$\{$ x''_i | $x_i \in C_c$ in the parent node of $LN\}$, as well as the references to these objects (i.e., encrypted objects in a server's database).

Each cluster C_c is defined by the cluster fingerprint CF_c that represents a compact information used for that cluster. On the other hand, each object x_i, either cluster object or outliers object, is represented by its binary fingerprint, x''_i, obtained by mapping the sparse object to a binary vector (as previously described in Section 5.1.2).

Each subspace cluster across the tree will have a different fingerprint that holds only the local relevant dimensions to that cluster (i.e., represented by 1 values). Compared with the methods in [60, 63], each object will have multiple-fingerprint representations from the root to its own fingerprint in the leaf node instead of the

full vector representation. Similar to the method presented in [60], these various representations allow to prune objects along the path of the tree and reduce the search space.

We do not store the actual objects or their fingerprints in the inner node clusters because we want to see the effect of hierarchical nesting clustering on the fingerprints indexing as well as reduce the number of evaluations for the fingerprint query. However, the references (i.e., indices or pointers) to the objects are used only for clustering purposes.

Candidate Pruning In existing multi-level indexes [60, 63], the leaf nodes usually store the full representations for the objects. This representations help in pruning the whole subtree of the index and minimizes the search space. Given a query object q, the lower bounding distance for all objects within the underlying subtree is well defined; the distance between q and any inner node in the subtree, defined by their related dimensions, should be smaller than the distance to each object in the leaf node in the same subtree.

In our index, we use the reduced dimensions information, performed by the fingerprinting process, not only in the inner node for the clusters, but also in the leaf node for the objects. To achieve the lower bounding property in this case, we strictly set the number of the sparsified dimensions in the inner nodes to be larger than the those for the leaf nodes during the index construction. Formally, let assume that the number of relevant dimensions to an inner node clusters, defined in CF, is set to be $\mid CF \mid = m_I$ (i.e., the cardinality of the cluster fingerprints or the number of 1's), then the number of the relevant dimensions for each object fingerprint, x'', in the leaf node, defined as $\mid x'' \mid = m_L$, should be larger than m_I ($m_L > m_I$).

Furthermore, based on the techniques used for the fingerprinting in our index, it is expected that the cluster fingerprint shared most of the relevant dimensions (1's

values) with its leaf node fingerprints; the relevant dimensions for the cluster are derived based on the aggregated information of the objects in that cluster. That is, $CF \subset x''$.

Figure 5.2 shows an example of the tree structure of the LID-Fingerprint. The number of relevant dimensions for the inner node, m_I, is set to 2, while for the objects in the leaf node and the outliers list, the relevant dimensions, m_L, is set to 4. In the root, two subspace clusters are generated, each represented by its own fingerprint that mainly represents the relevant dimensions to its cluster, as well as a set of outliers fingerprints {5,9}. The root first cluster's objects are further clustered into sub-clusters, with a different fingerprint represents each sub-cluster. Cluster 2 in the root, however, is appended with the leaf node that contains the cluster object fingerprints {3,7,9, ..}. Similar to SUSHI and $\triangle^+$-tree, it is clear that LID-Fingerprint constructs unbalanced tree, where some objects may have multiple-cluster fingerprint representations, while other have only one or few.

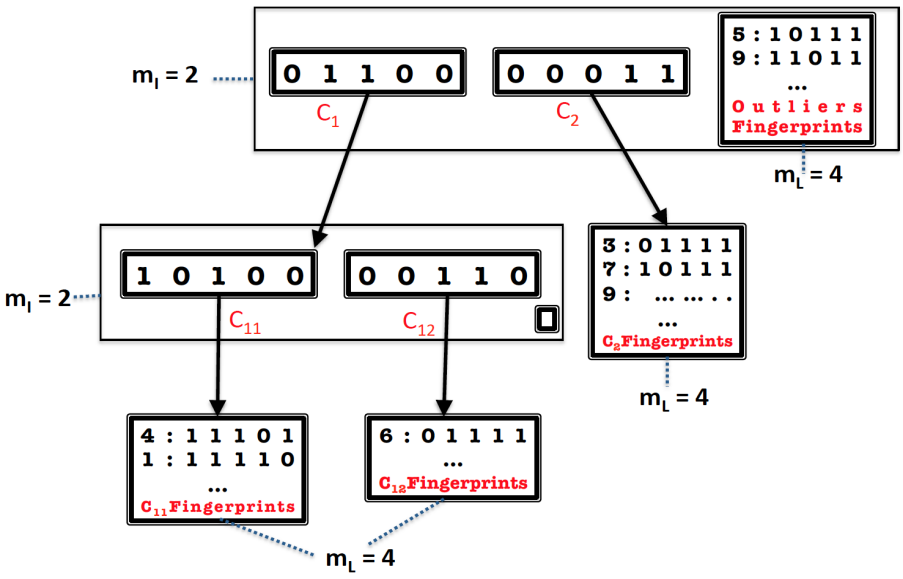


Figure 5.2 Multi-level index structure of LID-Fingerprint.

Clustering Evaluation Since k-LIDoids subspace clustering depends on a random initialization to generate an initial form of the clusters, which then enhanced gradually using the feature ranking and sparsification techniques, it might not be guaranteed to obtain the best quality clustering from one execution. Datasets usually can have multiple clustering which can be exposed by multiple executions for the clustering algorithm with different relevant dimensions per cluster [60]. Moreover, there is no guarantee that the relevant dimensions or the clusters in one execution will be exactly the same to another execution. This variation in the generated clusters may lead to a low search performance in the generated index [60]. Therefore, it is important to evaluate the clustering results of multiple executions, and select the best quality clusters for each inner node in the index.

Computing the fingerprint for a cluster or an object mainly depends on the nearest neighbors for that object or the cluster representative m_c (as previously presented in Sections 3.2.1 and 4.1.3). Thus, for evaluating the clusters in each inner node, we select a measure that reflects the connectedness of the cluster partitions. The connectedness relates to what degree objects and their nearest neighbors are placed in the same cluster in the clustering results, and it is evaluated by the connectivity measures[154, 155] as follows: Let NN_{i_j} be the jth nearest neighbor of object x_i, and let $y_{(i,NN_{i_j})}$ has a value of 0 if x_i and NN_{i_j} are placed in the same cluster and $1/j$, otherwise. Then, for a particular clustering configuration, $subspace\_C$, without considering the outliers set, the connectivity is defined as,

$$conn(subspace\_C) = \sum_{i=1}^{n} \sum_{j=1}^{p} y_{(i,NN_{i_j})} \tag{5.1}$$

where n is the size of the dataset, and p is the neighborhood size in the p-NN graph entries used for both NNWID-Descent and k-LIDoids. The total value of conn(subspace_C), which can have a value in the range of $[0, \infty]$, should be minimized [154, 155].

Fingerprinting and Indexing Algorithm To construct the multi-level index (Algorithm 6), we start with clustering the full dataset X multiple times, using k-LIDoids, until we get the best clustering quality, which is measured by the least connectivity value. The fingerprints of these resulted clusters are stored in the root, and the cluster object references are held temporary for the next recursive sub-clustering. For each cluster in the root, k-LIDoids is applied several times on the subset of objects of that cluster, and the best quality sub-clusters are selected (lines 9-17). These sub-cluster fingerprints are stored in an inner node (line 18), which is then attached to the parent node (i.e., the root). However, if the number of objects in any cluster is below a specific threshold, a leaf node is created to store the fingerprints for those objects, which are computed using the objects fingerprinting process, and attached to that cluster's fingerprint (lines 1-4). This process is repeated until the index is complete. Since the object fingerprinting process is independent process used for computing the object fingerprints, it is worth to mention that for efficiency, object fingerprints can be computed initially before building the index, and stored in the leaf nodes once they are created.

5.1.4 Nearest Neighbor Search Process

Similarity Measure A common metric used in the literature to compare binary vectors (binary fingerprints) is the Hamming distance $\mathcal{H}$, which can be computed using a bitwise XOR followed by a bit count. We use a slightly modified version of the Hamming distance, called the subspace Hamming distance, $\mathcal{H}_s$, to focus only on the subspace or the relevant dimensions to the object fingerprints. For example, the subspace Hamming distance $\mathcal{H}_s$ between a query object fingerprint, denoted as x''_q, and any object fingerprint, x''_i is computed as a bitwise AND operation between x''_q and the complement of x''_i, $\bar{x}''_i$, followed by a set bit count on the result. Formally,

$$\mathcal{H}_s(x''_q, x''_i) = \sum_{j=1}^{D} (x''_{qj} \& \bar{x}''_{ij}), \tag{5.2}$$

Algorithm 6: Multilevel Index Construction Method

```
Input  : Dataset X, the number of clusters k, sparsification rate for inner node m_I, sparsification rate
         for leaf node m_L, p-nearest neighbor size
Output: Node N
if |X|≤ minimum_size then
    Fingerprints[ ]= Fingerprinting_Process(X,p,m_L);
    Return Construct_Leaf_Node(Fingerprints)
end
Best_Clusters=Subspace_Clustering_using_k-LIDoids(X,k, p, m_I);
Best_Outlier_Fingerprints[ ]=
Generate_Fingerprints_using_NNWID_Descent(Best_Clusters.Outliers, p,m_L);
Best_Connectivity=Evalaute_Clustering(Best_Clusters);
for maximum_number_of_executions (execute in parallel) do
    Current_Clusters=Subspace_Clustering_using_k-LIDoids(X,k, p, m_I);
    Current_Outlier_Fingerprints[ ]=
    Generate_Fingerprints_using_NNWID_Descent(Best_Clusters.Outliers,p,m_L);
    Current_Connectivity=Evaluate_Clustering(Best_Clusters);
    if Current_Connectivity < Best_Connectivity then
        Best_Clusters=Current_Clusters;
        Best_Outlier_Fingerprints[ ]=Current_Outlier_Fingerprints[ ];
        Best_Connectivity=Current_Connectivity;
    end
end
Node Inner_node=Construct_Inner_Node(Best_Clusters.Fingerprints, Best_Outlier_Fingerprints);
foreach cluster C_c ∈ Best_Clusters (execute in parallel) do
    Run Multilevel Index Construction Method for C_c.objects
end
Return inner_node;
```

where & denotes the bitwise AND operation. $\mathcal{H}_s$ helps to minimize the distance between x''_q and x''_i, if and only if x''_i is a related object to x''_q; x''_i shares the same relevant dimensions to x''_q, regardless to other relevant dimensions defined for x''_i.

Searching Algorithm The complete pseudo-code for the search process is given in Algorithm 7, which basically follows the searching method in [60, 63], but with using fingerprints and the $\mathcal{H}_s$ distance function. Given a query object q, the fingerprint, x''_q, of q is computed by simply mapping each non zero feature to 1 (Section 5.1.2).

To search for the K-NN nearest neighbor fingerprints using LID-Fingerprint index, we create two data structures, a priority queue, Q, to hold the current active node with its $\mathcal{H}_s$ distance from x''_q, and an array list, *Results*, to hold the K-NN nearest neighbor candidates in ascending order of their distances to x''_q. The distance between the Kth fingerprint and x''_q is used to prune a way the search space.

We define a variable Max-Dist, which is initialized to ∞, and works as a pruning distance as it will be described later. Starting from the root node, which get inserted into Q, we repeat the operations in lines (7-22) until Q becomes empty. The first node is pulled from the Q, and tested. If the node is a leaf node, the distances between its fingerprints and x''_q are computed using a linear scan search (11-16). The fingerprints that closet to x''_q are used to update the *Results* list. The value of the pruning distance, Max-Dist, is updated with the K-NN distance (if it exists). Otherwise, the node is an inner node, and its outliers fingerprints (if exists) are checked first using a linear scan and the results are inserted into the *Results* list. Then, for each sub-cluster fingerprint, the distance from x''_q to that sub-cluster fingerprint is computed. If this distance is less than Max-Dist, then the node that is connected to that cluster fingerprint is inserted to the queue Q. If the distance from x''_q to any cluster fingerprint in any inner node is larger than the current Max-Dist, then we prune the subtree branch of that cluster. We assume that the Kth distance is less than any object fingerprints derived from that cluster because these fingerprints have many more dimensions (with values=1) in compared with the cluster fingerprint (as we described in Section 5.1.3).

5.1.5 Updating the Index

In this section, we explain the possible procedure that can be used for updating already created index in terms of inserting a new fingerprint or deleting existing one.

Algorithm 7: K-NN Search

```
Input  : query object q, the nearest neighbors size K, index Tree
Output: K-NN results
1  Fingerprint x''_q= Map_Query_Object_to_Fingerprint(q);
2  Q=new Priority Queue(); (to store the a list of (node, distance))
3  KNN_Results= new List(size K); (to store the list of (fingerprint, distance) in ascending order distance)
4  Max_Dist=∞;
5  Current_Fingerprints=new List();
6  Q.insert(Tree.root, 0.0);
7  while Q is not empty and Q.Get_First_Distance ≤ MAX_Dist do
8      Node N=Q.Poll_First_Node();
9      if N is a leaf node then Current_Fingerprints= N.Fingerprints; ;
10     else Current_Fingerprints= N.Outliers_Fingerprints;;
11     foreach Fingerprint x'' in Current_Fingerprints do
12         if H_s(x''_q, x'') ≤ Max_Dist then
13         KNN_Results.add(B, H_s(x''_q, x''));
14         KNN_Results.sort();
15         Max_Dist=KNN_Results.get(K).distance ;
16     end
17     if N is a inner node then
18         foreach Cluster_Fingerprint CF_c in N do
19             if H_s(x''_q, CF_c) ≤ Max_Dist then Q.insert(N.CF_c.Node);
20         end
21     end
22 end
23 Return KNN_Results;
```

Insertion Algorithm 8 describes the insertion operation for the LID-Fingerprint index. When a new object x_{New} has to be inserted to the index, its fingerprint, x''_{new}, is computed (line 1) (using object fingerprinting process) and inserted to the appropriate leaf node in the index. The Index is traversed from the root down to the leaf nodes by selecting the closet sub-cluster fingerprint along the path using $\mathcal{H}_s$ as a distance measure (line 3-10). However, depending only on the distance may not guarantee that the appropriate leaf node will be selected for the new object. Therefore, for each inner node, it is important also to check if the closest sub-cluster fingerprint shares most of its relevant dimensions with the new fingerprint, x''_{new} (say

95% of the relevant dimensions). Otherwise, the new object fingerprint will be inserted to the outliers list of the current active inner node. If the nearest leaf node is full, then k-LIDoids clustering has to be applied on that leaf node objects, which will introduce a new inner node in the tree.

However, the insertion operation may require to periodically reconstruct the index using the original dataset regardless of the index performance to cope with many inserted fingerprints in the tree [63].

Algorithm 8: Insertion Operation

```
Input : new object x_New, Index Tree
Fingerprint x''_New= Find_New_Object_Fingerprint(x_New); (Fingerprint is generated using object
fingerprinting process Strategy);
Node N=Tree.root;
while N is not a leaf node do
    Closest_CF= N.CF_1;
    foreach Cluster_Fingerprint CF_c in N do
        if H_s(x''_New, CF_c) ≤ H_s(x''_New, Closest_CF) then Closest_CF=CF_c ;
    end
    if Closest_CF ⊂ x''_New then N=Closest_CF.Node; ;
    else N.insert_to_Outliers_Fingerprints_List(x''_New); Break ;
end
if N is a leaf node then
    N.insert_New_Fingerprint(x''_New);
    if N is full then
        Run Multilevel Index Construction Method for N objects using N.Fingerprint references
    end
end
```

Deletion Similar as the insertion operation, to delete an object from the tree, the object fingerprint is computed using the object fingerprinting process. For each inner node, this fingerprint is initially checked against the outliers list before checking any of the sub-cluster fingerprints of that node. If it is not found in the outliers list, the object fingerprint is checked along the path from the root to the leaf node to find the closest sub-cluster fingerprint. In the leaf node, the object is searched linearly and

deleted. If the leaf node size become less than a defined threshold, then the sub-cluster fingerprint of that leaf node (which is one level above the leaf node) is deleted, and its fingerprints are added to the outliers list in the inner node that contained the sub-cluster fingerprints. Similar as the insertion process, if many deletion operations were performed, it is necessary to rebuild the index using the original dataset.

5.1.6 Information Hiding Aspects of LID-Fingerprint

Two primary information hiding techniques are included in LID-Fingerprint: *data suppression*, and *data masking*. Fingerprinting in general can make these techniques more valuable in terms of protecting the identities, privacy, and personal information by not releasing—to semi-honest users—the actual values of some of the dataset information; these values that may lead to infer some of the sensitive information. While some of dataset information (mostly public) is entirely removed in *data suppression*, *data masking* is the process of concealing or encrypting the selected information. The masked data remains encoded in the database and can be accessed or re-identified by only authorized persons.

With using feature ranking and sparsification processes in both k-LIDoids and NNWID-Descent, LID-Fingerprint includes *data suppression* by removing many noisy features locally from each object. *Data masking* is also involved in LID-Fingerprint by mapping (encoding) the remaining important features to binary representations. This transformation for the feature vectors reduces the quality of datasets, and changes the overall statistics that causes the data to become practically useless for unauthorized observers. Beside the *data suppression* and *data masking* processes, LID-Fingerprint does not guarantee the uniqueness of the generated fingerprints; neighbor objects in a dataset may have exactly similar fingerprints. This makes K-NN search results completely anonymous, unless the unauthorized person (i.e., attacker) has a direct access to the actual values of the dataset.

5.2 Experimental Framework

For the comparison of LID-Fingerprint with other competing methods, we first conducted experiments to study the influence of using LID-Fingerprint's object fingerprints, instead of using the actual data, in the nearest-neighbors graph construction accuracy. Then, we tested the LID-Fingerprint similarity measure and indexing performance against other state-of-the-art methods.

5.2.1 Competing Methods

The performance of LID-Fingerprint is contrasted with four competitors:

- **Linear Scan**: In order to search for K-NN fingerprints, a brute-force method is used to compare the query fingerprint x''_q with every other fingerprints in the database. The fingerprints are generated by the LID-Fingerprint fingerprinting process (Algorithm 5) as described in Sections 5.1.2 and 5.1.2.

- **Min-Hash**: Min-Hash algorithm is a min-wise independent permutations locality sensitive hashing (LSH) scheme, designed for Jaccard similarity. Min-Hash allows producing similar signatures for fingerprints that have a high Jaccard similarity. A Min-Hash signature is a sequence of numbers produced by multiple hash functions h_l applied on a binary dataset.We also applied Min-Hash technique on the fingerprints that are generated by LID-Fingerprint fingerprinting process (Algorithm 5) in order to test the performance of existing indexing method on these fingerprints.

- **Super-Bit LSH**: As LID-Fingerprint, Super-Bit involves both fingerprinting and indexing processes. Super-Bit improves the random projection in LSH by computing an estimation of cosine similarity. Super-Bit divides the random projections into A groups, which are then orthogonalized in B batches of G vectors. Thus, we obtain B G-super bits for each group. G is called the Super-Bit depth, B is the number of Super-Bits, and A= $G * B$ is the Super-bit code length.

- **SUSHI**: is a general framework for a multilevel index structure based on a hierarchal nesting top-down subspace clustering (as described in Section 2.2.3). For a fair comparison with LID-Fingerprint, we tested SUSHI using k-LIDoids a subspace clustering algorithm instead of PROCLUS [156] and MINECLUS [157] that are used by the authors in [60]. We also used the clustering connectivity for the clustering evaluation as the evaluation method used in [60] has a very high computational complexity. The rationale for the comparison with this method is to show what is the gain in terms of the time and the search accuracy behind using the binary representations (fingerprints) for the reduced dimensions instead of using the actual values for these dimensions.

5.2.2 Datasets

Seven real datasets of varying dimensions were considered:

- **ALOI-100** is subset of the Amsterdam Library of Object Images (ALOI) [135], which contains 110,250 images of 1000 small objects. Each image is described by a 641-dimensional feature vector based on color and texture histograms. ALOI-100 contains 10,800 images of 100 simple objects generated by selecting the objects uniformly from among the classes.

- **MNIST** [136] contains 70,000 images of handwritten digits. Each image is represented by 784 gray-scale texture values. MNIST is a combination of two of NIST's databases: Special Database 1 and Special Database 3 contain digits written by high school students and United States Census Bureau employees, respectively, with a total of 10 possible representations for the digits.

- **RLCT** The Relative Location of CT dataset (RLCT) [138] contains 53,500 axial CT slice images from 97 different patients. Each CT slice is described by two histograms in polar space. The feature vectors of the images are of 385 dimensions.

- **Wearable Computing** Classification of Body Postures and Movements (PUC-Rio) dataset [145] contains 165,633 samples collected on eight hours of activities of four healthy subjects in different static postures and dynamic movements. Each sample's features vector has 18 attributes that represent user data such as name, gender, age, height, weight, body mass, and sensor axis values. There are five possible positions (sitting-down, standing-up, standing, walking, and sitting).

- **Online News Popularity dataset (ONP)** [146] contains 39,644 articles' textual extracted data, each has 60 attributes (58 predictive attributes, and 2 non-predictive) that describe different article aspects. The articles are binary classified as popular and unpopular using a decision threshold of 1400 social interactions.

- **MiniBooNE** [138] particle identification dataset contains 130,065 of signal and background events Each event has 50 particle ID variables. This dataset is taken from the MiniBooNE experiment and is used to distinguish between two classes, electron neutrinos (signal), and muon neutrinos (background).

- **Sensorless Drive Diagnosis dataset (SDD)** [138] includes 58,509 electric drives, with 48 features extracted form electric current drive signals. The drive has intact and defective components. This results in 11 different classes with different conditions

5.2.3 Accuracy of LID-Fingerprint based on Nearest Neighbors Graph Construction vs. Sparsification

In this experiment, we compared between the p-NN graph that is updated in each iteration of NNWID-Descent, after sparsifying a Z portion of the features in each

dataset object, and the p-NN graph constructed using the fingerprints generated by LID-Fingerprint's object fingerprinting process. The fingerprints exact p-NN graph is computed using the similarity measure defined in Section 5.1.4. We run the experiment in all datasets to show the effect in different dataset sizes and dimensions.

Parameters Setting While the nearest neighbors size p in the p-NN graph is set to 10 for NNWID-Descent, the LID-Fingerprint graph is computed for p set to 10, 50, and 100. The proportion of the sparsified features, Z, in both NNWID-Descent and LID-Fingerprint fingerprinting algorithms, is varied with different datasets as it depends heavily on the density of the feature vectors. For example, in each iteration, we set Z to 0.02 for ONP, 0.04 for MiniBooNE and SDD, 0.055 for Wearable Computing, and 0.0025 for the sparse datasets, MNIST and RLCT, and ALOI-100. Additionally, the nearest neighbors size, K, that used for computing the feature's support-weighted ID score, wID, is set to be 100 for all datasets. The number of iterations T, is varied according to the dataset dimensions. T is set to 40 for MNIST, 70 for RLCT and ALOI-100, 29, 20, 18, and 15 for ONP, MiniBooNE, SDD, and Wearable Computing, respectively.

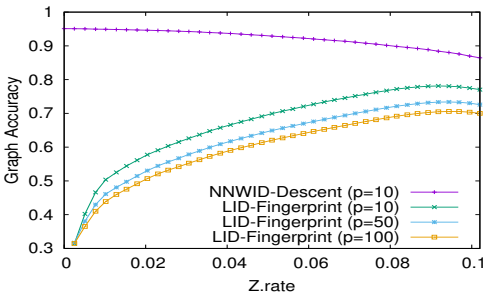
Performance Measure For each of the considered datasets, the graph accuracy is used as a performance measure. The class labels of the data objects were used to measure the quality of the resulting p-NN graph at every iteration. The accuracy of the resulting p-NN graph is evaluated, as in [18], using the following formula:

$$graph\ accuracy = \frac{\#correct\ neighbors}{\#data \times p}, \tag{5.3}$$

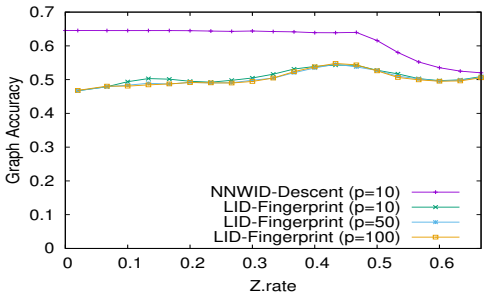
where the 'correct' neighbors share the same label as the query object.

Results and Analysis Figure 5.3 shows the p-NN graph accuracy at every iteration for both NNWID-Descent and LID-Fingerprint graphs. For each dataset, except

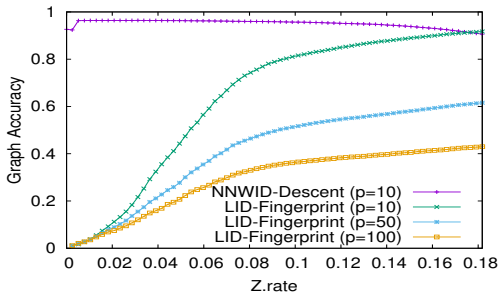
for Wearable Computing dataset, we notice that when the number of the sparsified features increases, the fingerprints graph accuracy improves and becomes closer to NNWID-Descent graph accuracy (with a gab range from [0-$\leq$ 20%]). However, there might be a little degradation in the accuracy of NNWID-Descent graph. We conclude that the generated fingerprints, using LID-Fingerprint process, can have some uniqueness among nearest neighbor objects. This allows us to use those fingerprints, instead of the original or sparse data, for the nearest neighbor similarity search, which is faster to compute using binary operations, and more secure as those fingerprints do no reveal any feature values. Wearable Computing dataset shows almost no changes in the performance of fingerprints graph accuracy— with a gab difference $\geq$ 30% between the fingerprints graph and NNWID-Descent graph accuracy— as its number of dimensions is relatively too small to generate unique fingerprints among the neighbor dataset objects.



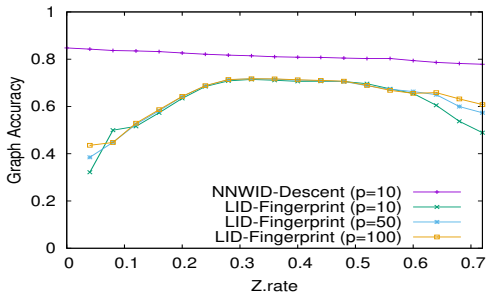
(a) MNIST



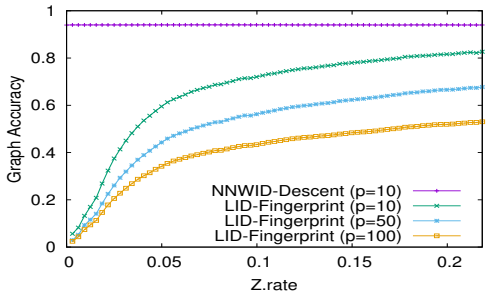
(b) ONP



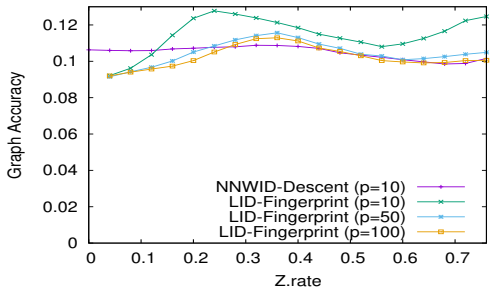
(c) RLCT



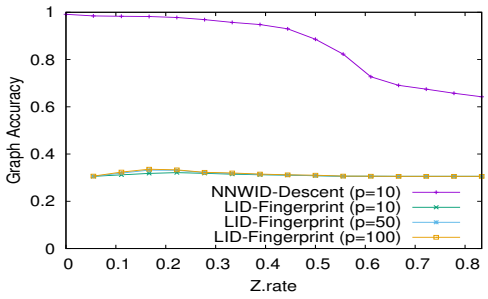
(d) MiniBooNE



(e) ALOI-100



(f) SDD



(g) Wearable Computing

Figure 5.3 Comparison between NNWID-Descent graph (p=10) and LID-Fingerprint fingerprints graph with different values of p (10, 50, and 100).

5.2.4 Comparison of LID-Fingerprint with its competitors

In this experiment, we tested the performance of the indexing and the similarity measure, in terms of K-NN search, of LID-Fingerprint in compared with the other competing methods. The experiment includes two parts: First, LID-Fingerprint index is compared with the linear scan search, Min-Hash and Super-Bit, as fingerprinting search and indexing methods. Then, we compared between LID-Fingerprint index and SUSHI to see the gain of using fingerprints in multi-level indexing technique instead of actual feature vectors. While the linear scan is considered an exact search for the fingerprints, other methods are considered an approximate search as they start by indexing (i.e SUSHI and LID-Fingerprint) or hashing (i.e Min-Hash and Super-Bit) the dataset objects prior applying the K-NN search. The query object q is randomly selected from each dataset. In LID-Fingerprint indexing, Linear scan, and Min-Hash, the corresponding fingerprint x''_q of q is selected from the fingerprints generated by the fingerprinting process (Algorithm 5). On the other hand, the query object is selected from the actual dataset in Super-Bit and SUSHI methods. For each method and each of the dataset considered, we randomly selected 1000 objects to serve as queries, and the best 20-NN matched objects to a given query are obtained. The experiment results are averaged for each parameter setting.

Performance Measure Three evaluation parameters are measured: the average query accuracy, the average number of distances, and the average search time. The number of distances equals the number of target distance evaluations needed by the search process to return the query result. The search time is shown as a proportion of the time in milliseconds needed to return the query's K-NN result. For one query q, the accuracy of its K-NN result is defined as:

$$query\ accuracy = \frac{|\{x''_i | y_{x_i} = y_q\}|}{K}, \tag{5.4}$$

where y defined as a true label for the object. The query accuracy measures the ratio of fingerprints that have the same label as the query object.

Fingerprinting Search and Indexing Comparison

Parameters Setting For the LID-Fingerprint indexing, the linear scan, and Min-Hash, the fingerprints set is generated (by the object fingerprinting process) using the parameter setting of Z and T specified in Table 5.1 for each dataset, where these values are selected according to the highest fingerprint graph accuracies in Figure 5.3. The nearest neighbors size, p, used to find the features wID values is set to 100.

In LID-Fingerprint, the number of clusters, k, for k-LIDoids clustering in each inner node, is fixed to be equal to the ground truth labels for all datasets. In ALOI-100 and RLCT, however, the number of clusters is set to 10, in order to avoid the large computational time associated with the large number of the ground truth labels (i.e 100 and 97, respectively). The minimum_size is set to be $Min(n/k, 1000)$, where n is the number of dataset objects (assuming that the objects are normally distributed among the ground truth labels). The outliers-threshold is set to 5, and the maximum_number_of_executions for the subspace clustering for each inner node, to find the best quality clusters, is set to 5 as well. The number of iterations M and the features sparsification rate Z for k-LIDoids are varied according to the dataset dimensions. However, we set M strictly larger than T, in order to ensure that the cardinality of cluster fingerprints is less than the cardinality of object fingerprints. More precisely, using M, T, and Z, the clusters final cardinality value, m_I, in the inner nodes, is computed as $m_I = D - (Ceil(D * Z) * M)$. Similarly, the objects cardinality value, m_L, in the leaf nodes, is computed as $m_L = D - (Ceil(D * Z) * T)$, where $M > T$, and $Ceil$ is the ceiling function. This will increase the gab between m_L and m_l, and ensure that $m_I < m_L$. Table 5.1 shows the M, T, Z, m_L, and m_I setting for all datasets.

For Min-Hash and Super-Bit methods, we set the number of hash functions to be equal to 2 for the datasets with a small number of dimensions, D, such as in ONP, SDD, Wearable Computing, and MiniBooNE. For a large value of D (i.e., MINIST, RLCT, and ALOI-100), the number of hash functions is set to 10. For all datasets, the number of buckets is set to be equal to the ground truth labels, except for ALOI-100 and RLCT, where it is set to be equal to 10.

Table 5.1 Parameter Setting for all Datasets: the Number of Iterations M for k-LIDoids, the Number of Iterations T for the Object Fingerprinting Process, the Features Sparsification rate Z, and the Cardinality Values for the Clusters and Objects Fingerprints, $m_I = D - (Ceil(D * Z) * M)$ and $m_L = D - (Ceil(D * Z) * T)$, Receptively

Datasets	D	Z	M	T	m_L	m_I
ALOI-100	641	0.0025	67	62	517	507
MNIST	784	0.0025	45	40	704	694
RLCT	385	0.0025	65	60	325	320
Wearable Computing	18	0.055	13	8	10	5
ONP	60	0.02	17	14	32	26
MiniBooNE	50	0.04	15	12	26	20
SDD	48	0.04	18	15	18	12

Results and Analysis Figure 5.4, (a) through (c), shows the average time, the average number of distance, and the average query accuracy for the competing methods. For all datasets, it can be seen that the linear scan search outperforms other methods in terms of the query accuracy (Figure 5.4 (a)), which is expected since it is an exhaustive search. Except for ONP and RLCT, the LID-Fingerprint indexing outperforms or shows a comparable performance with Min-Hash. However, even that Super-Bit and Min-Hash have acceptable performance for some of the datasets, they have very low accuracy with the high-dimensional datasets such as ALOI-100 and

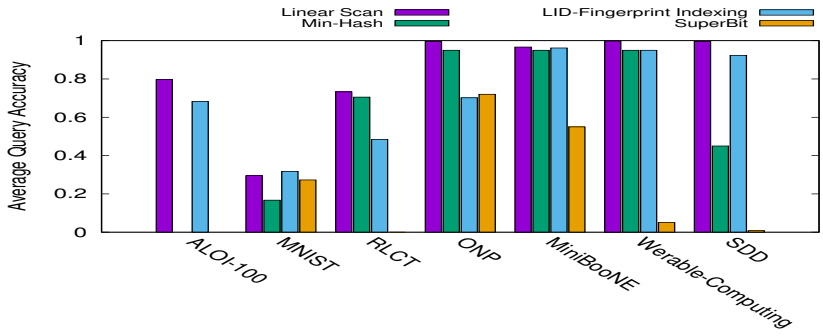
MNIST. Super-Bit, however, shows the least performance in most of the datasets, especially for SDD, RLCT, and Wearable Computing, with an average accuracy is < 0.2. In all competing methods, in the datasets with a large D, we noticed some failed queries, which are the queries with 0.0 accuracy. Even that these types of queries were ignored in this experiment; they are considered good sources for analyzing the possible causes of failed search.

Figure 5.4 (b) shows that the linear scan and the LID-Fingerprint indexing have the least time complexity in compared with other methods, which means that the similarity measure defined, the subspace Hamming distance, is more effective and efficient to compute, when compared to Jaccard and cosine similarities. However, while Min-Hash has the longest computational time (on average $>= 200ms$) for all datasets, Super-Bit also shows degradation in the time performance for MNIST and RLCT.

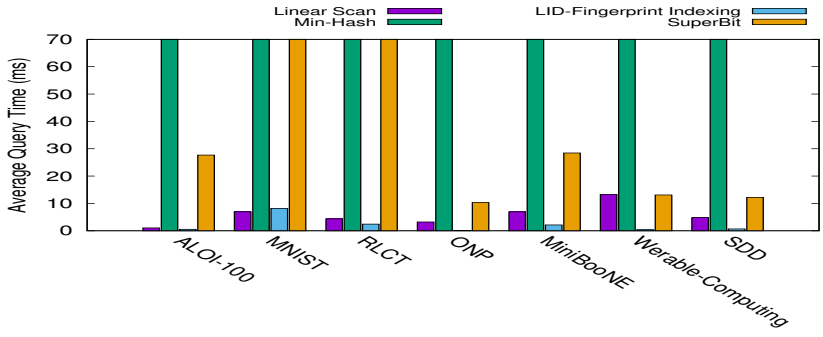
As shown in Figure 5.4 (c), the linear scan search has the largest number of average distance computations, which is essentially equals to the size of the datasets. On the other hand, in small-dimensional datasets, the LID-Fingerprint indexing shows a comparable performance with Super-Bit. However, this average performance relatively increases for the LID-Fingerprint indexing when it is applied on the datasets with a large D, such as ALOI-100, MNIST, and RLCT. In general, the number of distance computations of the LID-Fingerprint indexing depends on the number of generated clusters, the number of levels in the index, and the number of fingerprints in each cluster. Min-Hash has an opposite behavior of the LID-Fingerprint indexing; Min-Hash average distance computations increases for the datasets with a small D. Super-Bit, on the other hand, has the least number of average distance computations among all methods.

From all the above, we can see that there is a trade-off between the average distance computations, and the average time and accuracy for the K-NN search.

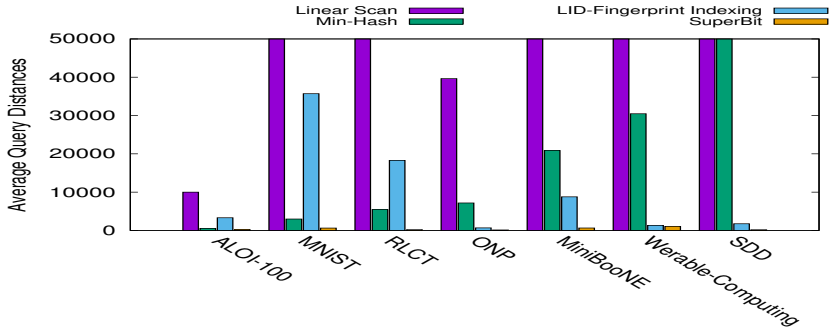
Even that the linear scan has the largest distance computations but it has the highest accuracy and the least computational time. On the contrary, Super-Bit has the least average distance computations with relatively long time and low accuracy.



(a) Query Accuracy



(b) Query Time



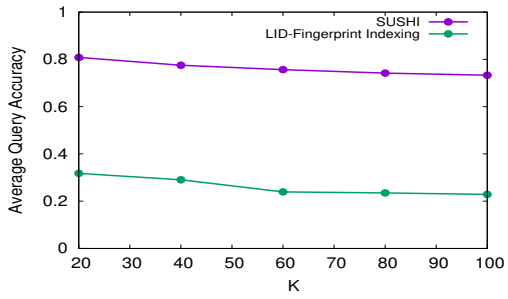
(c) Query Distances

Figure 5.4 The 20-NN search performance, in terms of the average accuracy, distance, and time, among the competing methods. For all datasets, the Min-Hash's time is $> 200ms$. The number of distance evaluations of the linear scan is equal to the size of the datasets.

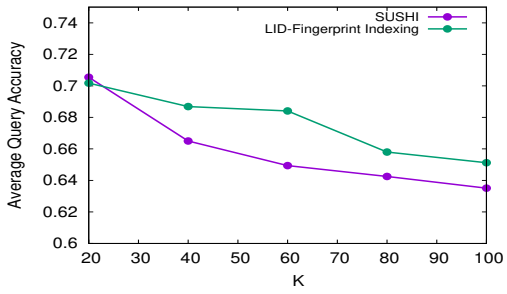
Comparison of LID-Fingerprint vs. SUSHI

Parameters Setting In SUSHI, we used the same parameters setting as the LIDFingerprint indexing, in terms of the number of clusters, k, the minimum_size, outliers-threshold, maximum_number_of_executions, the number of iterations M, and the features sparsification rate Z for k-LIDoids (which are specified in Table 5.1). We further considered five values of K for the search, where K=20,40, 60, 80, and 100. The query accuracy of the LID-Fingerprint indexing and SUSHI is evaluated for all K, but the time and distance performance are averaged among the different K, as we noticed that they only have slight increasing in values when the K increases.

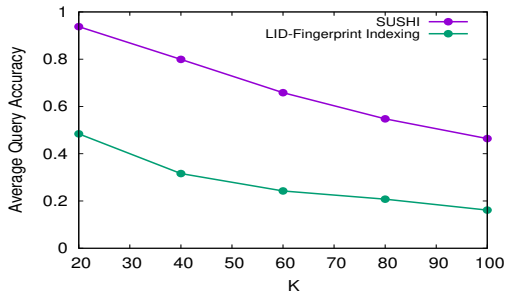
Results and Analysis For all datasets, Figure 5.5 shows the average query accuracy for both LID-Fingerprint indexing and SUSHI. We can see from the results that there is a gab between both methods ranges from 0% to 40%. This gab decreases–especially for the datasets with a small D–with increasing the value of K. However, increasing K in general means that the results may include many none-related objects to the query, which therefore decreases the overall average accuracy. To decrease the accuracy gabs between the two methods, specifically for large datasets, we need to increase the number of sparsified features of the objects in order to achieve more uniqueness among the generated object fingerprints.



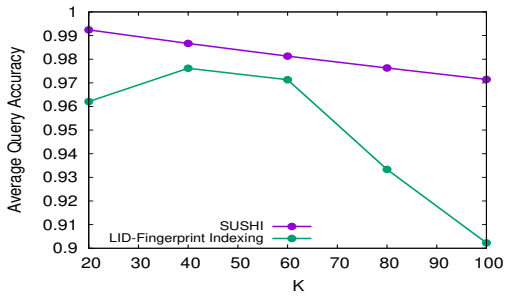
(a) MNIST



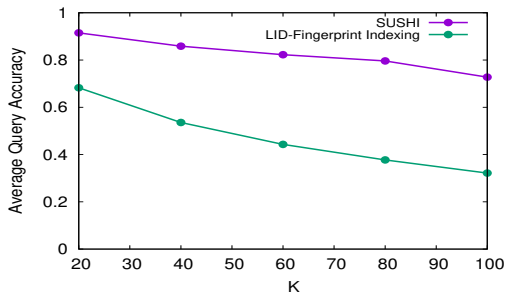
(b) ONP



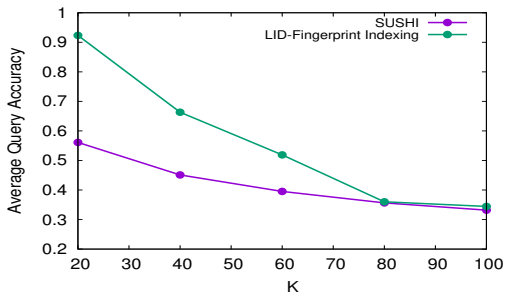
(c) RLCT



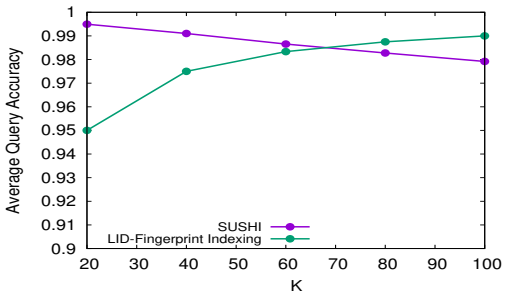
(d) MiniBooNE



(e) ALOI-100



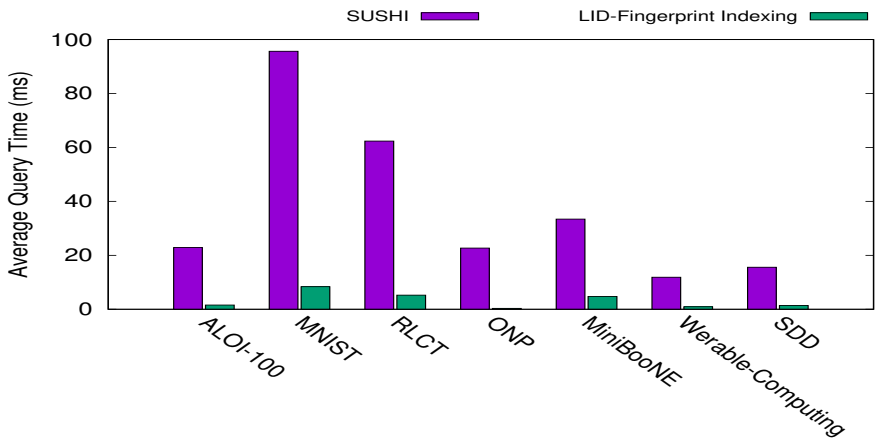
(f) SDD



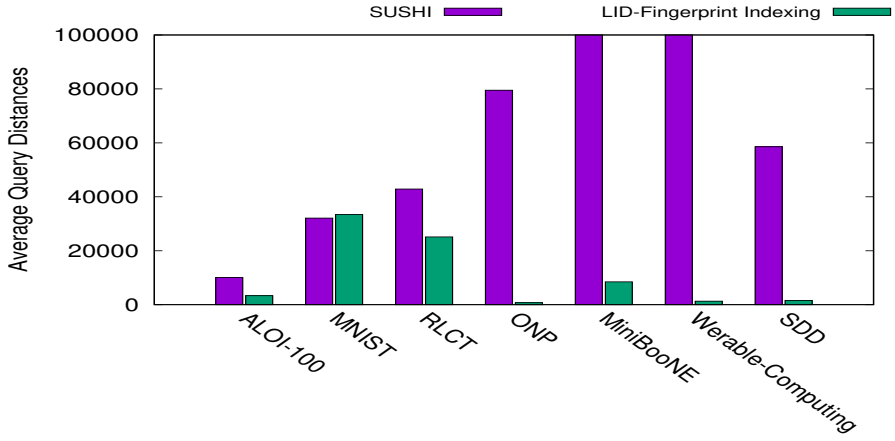
(g) Wearable Computing

Figure 5.5 Comparison between LID-Fingerprint indexing and SUSHI in terms of K-NN search with different values of K (20, 40, 60, 80, and 100).

Figure 5.6 shows the time and the distance evaluations for both methods. The LID-Fingerprint indexing has much less time performance than SUSHI, which is expected since the bitwise operations for computing the similarity between fingerprints are much faster than computing the regular Euclidean distance as in SUSHI. However, we also noticed that SUSHI has much larger number of distance evaluations when compared with LID-Fingerprint especially for the datasets with a small D. This is due to the fact that when the number of the relevant dimensions defined for each cluster is minimized, then using the lower bounding property for pruning the search space is no longer useful, and the K-NN search for the related objects to the query will expand to include all objects in the dataset. In general, when compared to SUSHI, LID-Fingerprint results are fairly acceptable since one of the main objectives of this research is hiding the information with achieving fast similarity search with a reasonable level of accuracy.



(a) Query Time



(b) Query Distances

Figure 5.6 The average search performance, in terms of time and distance evaluations, for the LID-Fingerprint indexing and SUSHI. For all datasets, the results are averaged over all values of K. The number of distance evaluations of SUSHI for MiniBooNE and Wearable Computing datasets is > 100000.

5.2.5 Preprocessing Time

For each dataset, we also show the preprocessing time (in seconds) for creating the index for all competitors (Table 5.2). We excluded the liner scan search as there is no index created other than generating the fingerprints using the LID-Fingerprint object fingerprinting process. For all methods considered in the comparison, we used

a Linux Server (Intel(R) Xeon(R) 2.70GHz) with eight cores, and with a memory size (16GB).

Table 5.2 Preprocessing Time (in Seconds) for all Methods Except the Linear Scan Search

Preprocessing time (in seconds)				
Datasets	*LID-Fingerprint*	*SUSHI*	*Min-Hash*	*Super-Bit*
ALOI-100	207908.803	215562.941	2.307	0.469
MNIST	28911.394	93247.733	20.630	3.456
RLCT	534119.168	587324.413	7.187	13.482
Wearable Computing	564768.957	568198.516	0.211	0.071
ONP	104734.760	95496.501	0.209	0.026
MiniBooNE	399675.664	710615.388	0.274	0.358
SDD	123895.192	107822.166	0.169	0.69

As expected, the LID-Fingerprint and SUSHI indexing techniques take longer preprocessing time as both are applied on actual data objects with all their dimension values. In contrast, the Min-Hash indexing is applied on previously created fingerprints. We noticed that the Super-Bit's preprocessing time highly depends on the number of hash functions used for the projection. Despite the long preprocessing time relative to its competitors, the LID-Fingerprint indexing shows a better potential improvement for secure similarity search. In general, the preprocessing time can be enhanced using more advanced High Performance Computing (HPC) servers, and/or the distributed computing techniques and algorithms.

5.3 Conclusion

In this chapter, we presented LID-Fingerprint as a new binary fingerprinting and indexing technique based on combining between NNWID-Descent and k-LIDoids algorithms. LID-Fingerprint fingerprinting process derives the fingerprints by mapping

the sparse representations for the data objects resulted from NNWID-Descent into binary representations. In large datasets, the number of generated fingerprints can be high and the similarity measure can be extensive to compute. Thus, we also developed a multi-level indexing data structure based on the subspace clustering algorithm, k-LIDoids, that allows reducing the number of computations and speeding up the search.

Using several real datasets, experimental results have shown that LID-Fingerprint can be applied to obtain binary fingerprints for high-dimensional feature vectors, as well as providing an efficient and secure indexing technique. When compared with other existing state-of-the-art methods, LID-Fingerprint can also provide a reasonable level of search accuracy. Beside the involved data suppression and data masking as data privacy protecting measures, LID-Fingerprint technique does not guarantee the uniqueness of the generated fingerprints, which is important for a reasonable level of data anonymity.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This dissertation mainly investigates the possibility of utilizing a new unsupervised feature selection criterion, Support-Weighted Intrinsic Dimensionality (support-weighted ID wID) [6]), in the design and analysis of search and clustering algorithms. Based on support-weighted ID, we have designed solutions for two of the data mining problems, including k-nearest neighbor graph construction and subspace clustering. Experimental results presented in this dissertation provide an evidence for the potential benefits of using support-weighted ID in improving the quality and performance of these algorithms, as well as achieving compact representations for the dataset objects. We further exploited the compact object representations in order to define a new binary fingerprinting and indexing framework for the high-dimensional data stored on the cloud.

To address the k-NN graph construction problem, we presented NNWID-Descent, a similarity graph construction method that iteratively improves k-NN graph accuracy using support-weighted ID while achieving a significant amount of sparsification of object feature vectors. NNWID-Descent can also be applied to obtain more compact representations for high-dimensional features vectors, which is important to reduce the storage and computational complexity for many applications. However, the ID estimator used in NNWID-Descent generally requires relatively large dataset sizes to provide a reasonable accuracy. Of the nine datasets used in our experiments, three are considered too small for the extreme-value-theoretic LID model to be applicable. Further improvement of NNWID-Descent could be achieved through the development of ID estimators that can more accurately handle smaller dataset sizes and smaller neighborhood sample sizes.

For clustering, we have presented k-LIDoids, a subspace clustering algorithm that exploits the use of support-weighted ID within k-medoids clustering to discover similar objects in a subset of features (subspace) of high-dimensional datasets. We have shown that our approach is able to achieve a better performance than the previous state-of-the-art subspace clustering algorithms. k-LIDoids is suitable for numerous applications that use high-dimensional datasets such as image segmentation, face clustering, and text and data compression. k-LIDoids can also be used to find compact clusters with few information stored for each cluster.

Basically, the limitations of k-LIDoids method follow the same limitations applied to the k-medoids clustering algorithm. To give examples to such limitations: First, k-LIDoids only discovers high-spherical compact clusters, but can not find other cluster shapes or dense clusters. Second, the algorithm also needs to determine in advance the initial number of clusters, k, which requires more information to be available in advance about the dataset that is being used, such as objects distribution and the desired clustering accuracy of the user. In future research, we intend to explore the use of support-weighted ID with other types of clustering algorithms (i.e., density and hierarchal clustering). Further investigation is needed as well for identifying the outliers cluster and the correlation between features in the cluster.

As a potential application of both NNWID-Descent and k-LIDoids, we presented LID-Fingerprint, a new binary fingerprinting and indexing framework based on using support-weighted ID. In LID-Fingerprint, the fingerprints are derived by mapping the sparse representations for the data objects resulted from NNWID-Descent into binary representations. LID-Fingerprint also includes a multi-level indexing data structure based on k-LIDoids in order to reduce the number of computations and speed up the search. The experimental study proved that our framework is able to generate binary fingerprints for high-dimensional datasets with an efficient and secure indexing technique compared to the state-of-the-art approaches. In addition to

providing two data privacy protecting measures: data suppression and data masking, LID-Fingerprint also provides a reasonable level of data anonymity.

LID-Fingerprint is important to provide secure search and reduce the storage and computational complexity for many cloud and smart-phone applications, and also can be integral to mobile device security. One direction of future research is by presenting a technique for filtering and refining the search results in the client end. Also, improving the LID-Fingerprint accuracy as well adopting a compression technique for providing more efficient and secure search are highly desirable as another potential future extensions to this work.

To conclude, support-weighted ID is utilized as the basis in the design of k-NN graph construction and subspace clustering algorithms. Specifically, support-weighted ID (wID) is used to dynamically guide the decisions of selecting the relevant features locally for dataset objects by providing a stable estimation for the contribution of each feature to the overall intrinsic dimensionality. Therefore, we consider wID to be highly important and more investigation should be made for integrating it within other learning models and applications beyond k-NN graph search and cluster analysis. Finally, the time complexity for the proposed algorithms or any other learning models based on wID can be enhanced using more advanced parallel and distributed computing techniques and algorithms.

BIBLIOGRAPHY

[1] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *International Work-Conference on Artificial Neural Networks*. Springer, 2005, pp. 758–770.

[2] I. K. Fodor, "A survey of dimension reduction techniques," Lawrence Livermore National Lab., CA (US), Tech. Rep., 2002.

[3] M. Dash and H. Liu, "Feature selection for clustering," in *Pacific-Asia Conference on knowledge discovery and data mining*. Springer, 2000, pp. 110–121.

[4] D. S. Modha and W. S. Spangler, "Feature weighting in k-means clustering," *Machine learning*, vol. 52, no. 3, pp. 217–237, 2003.

[5] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, no. 2, p. 559–572, 1901.

[6] M. E. Houle, "Local intrinsic dimensionality II: multivariate analysis and distributional support," in *International Conference on Similarity Search and Applications*. Springer, 2017, pp. 80–95.

[7] ——, "Dimensionality, discriminability, density and distance distributions," in *Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on*. IEEE, 2013, pp. 468–473.

[8] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K.-I. Kawarabayashi, and M. Nett, "Estimating local intrinsic dimensionality," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 29–38.

[9] D. Qin, S. Gammeter, L. Bossard, T. Quack, and L. van Gool, "Hello neighbor: Accurate object retrieval with k-reciprocal nearest neighbors," in *CVPR 2011*, June 2011, pp. 777–784.

[10] M. Brito, E. Chávez, A. Quiroz, and J. Yukich, "Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection," *Statistics and Probability Letters*, vol. 35, no. 1, pp. 33–42, 1997.

[11] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph," in *ICPR*, vol. 3, Aug 2004, pp. 430–433 Vol.3.

[12] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang, "Manifold-ranking based image retrieval," in *ACM MM*, 2004, pp. 9–16.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender systems – a case study," DTIC Document, Tech. Rep., 2000.

[14] W. Dong, C. Moses, and K. Li, "Efficient K-nearest neighbor graph construction for generic similarity measures," in *WWW*, 2011, pp. 577–586.

[15] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of Computational and Graphical Statistics*, vol. 15, no. 2, pp. 265–286, 2006.

[16] E.-H. Han, G. Karypis, and V. Kumar, "Text categorization using weight adjusted k-nearest neighbor classification," in *PAKDD*, 2001, pp. 53–65.

[17] Z. Wang and Z. Liu, "Graph-based KNN text classification," in *FSKD*, vol. 5, Aug 2010, pp. 2363–2366.

[18] M. E. Houle, X. Ma, V. Oria, and J. Sun, "Improving the quality of K-NN graphs through vector sparsification: application to image databases," *International Journal of Multimedia Information Retrieval*, vol. 3, no. 4, pp. 259–274, 2014.

[19] G. Nakhaeizadeh, *Industrial applications of data mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 479–480. [Online]. Available: https://doi.org/10.1007/BFb0094854

[20] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds., *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996.

[21] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, no. 2, pp. 139–172, Sep 1987. [Online]. Available: https://doi.org/10.1023/A:1022852608280

[22] K. Fukunaga, *Introduction to Statistical Pattern Recognition (2Nd Ed.)*. San Diego, CA, USA: Academic Press Professional, Inc., 1990.

[23] O. Zamir and O. Etzioni, "Grouper: a dynamic clustering interface to web search results," *Computer Networks*, vol. 31, no. 11, pp. 1361–1374, 1999.

[24] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. Citeseer, 2001.

[25] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[26] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[27] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992

[28] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[30] R. Vidal, "A tutorial on subspace clustering," 2010.

[31] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[32] D. M. Witten and R. Tibshirani, "A framework for feature selection in clustering," *Journal of the American Statistical Association*, vol. 105, no. 490, pp. 713–726, 2010.

[33] W. Gu, S. Dong, Z. Zeng, and J. He, "An effective news recommendation method for microblog user," *The Scientific World Journal*, vol. 2014, 2014.

[34] R. Agrawal, J. E. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," Dec. 14 1999, uS Patent 6,003,029.

[35] S. Goil, H. Nagesh, and A. Choudhary, "Mafia: Efficient and scalable subspace clustering for very large data sets," 1999.

[36] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, "Fast algorithms for projected clustering," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/304182.304188

[37] J. Yang, W. Wang, H. Wang, and P. Yu, "delta;-clusters: capturing subspace correlation in a large data set," in *Proceedings 18th International Conference on Data Engineering*, 2002, pp. 517–528.

[38] S. Katzenbeisser and F. Petitcolas, *Information hiding*. Artech house, 2016.

[39] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973. [Online]. Available: http://doi.acm.org/10.1145/362375.362389

[40] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding-a survey," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1062–1078, Jul 1999.

[41] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981. [Online]. Available: http://doi.acm.org/10.1145/358549.358563

[42] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Proceedings of the First International Workshop on Information Hiding*. London, UK, UK: Springer-Verlag, 1996, pp. 137–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=647594.731526

[43] J. Y. Halpern and K. R. O'Neill, "Anonymity and information hiding in multiagent systems," *Journal of Computer Security*, vol. 13, no. 3, pp. 483–514, 2005.

[44] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[45] D. Boneh and J. Shaw, "Collusion-secure fingerprinting for digital data," *IEEE Transactions on Information Theory*, vol. 44, no. 5, pp. 1897–1905, Sep 1998.

[46] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 41, no. 3, pp. 271–284, 2005.

[47] 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*. IEEE, 2006, pp. 459–468.

[48] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[49] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *null*. IEEE, 2003, p. 750.

[50] K. Moravec and I. J. Cox, "A comparison of extended fingerprint hashing and locality sensitive hashing for binary audio fingerprints," in *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*. ACM, 2011, p. 31.

[51] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," 2008.

[52] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ser. ICCV '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 750–. [Online]. Available: http://dl.acm.org/citation.cfm?id=946247.946721

[53] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969 – 978, 2009, special Section on Graphical Models and Information Retrieval. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888613X08001813

[54] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 1753–1760. [Online]. Available: http://papers.nips.cc/paper/3383-spectral-hashing.pdf

[55] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, June 2005, pp. 524–531 vol. 2.

[56] F. Farooq, R. M. Bolle, T. Y. Jea, and N. Ratha, "Anonymous and revocable fingerprint recognition," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, June 2007, pp. 1–7.

[57] A. Nagar, S. Rane, and A. Vetro, "Privacy and security of features extracted from minutiae aggregates," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, March 2010, pp. 1826–1829.

[58] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, "Fig: Automatic fingerprint generation," in *Network and Distributed System Security Symposium*, 2007.

[59] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671516

[60] S. Günnemann, H. Kremer, D. Lenhard, and T. Seidl, "Subspace clustering for indexing high dimensional data: a main memory index based on local reductions and individual multi-representations," in *Proceedings of the 14th International Conference on Extending Database Technology*. ACM, 2011, pp. 237–248.

[61] K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: A new approach to indexing high dimensional spaces," in *VLDB*. Citeseer, 2000, pp. 89–100.

[62] H. T. Shen, X. Zhou, and A. Zhou, "An adaptive and dynamic dimensionality reduction method for high-dimensional indexing," *The VLDB Journal*, vol. 16, no. 2, pp. 219–234, 2007.

[63] B. Cui, B. C. Coi, J. Su, and K.-L. Tan, "Indexing high-dimensional data for efficient in-memory similarity search," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 3, pp. 339–353, 2005.

[64] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, 2018.

[65] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1, pp. 273 – 324, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437029700043X

[66] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of relieff and rrelieff," *Machine Learning*, vol. 53, no. 1, pp. 23–69, Oct 2003. [Online]. Available: https://doi.org/10.1023/A:1025667309714

[67] L. Song, A. Smola, A. Gretton, K. M. Borgwardt, and J. Bedo, "Supervised feature selection via dependence estimation," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 823–830. [Online]. Available: http://doi.acm.org/10.1145/1273496.1273600

[68] F. Marcelloni, "Feature selection based on a modified fuzzy c-means algorithm with supervision," *Information Sciences*, vol. 151, pp. 201 – 226, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025502004024

[69] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "A simultaneous feature adaptation and feature selection method for content-based image retrieval systems," *Knowledge-Based Systems*, vol. 39, pp. 85 – 94, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705112002924

[70] W. Jiang, G. Er, Q. Dai, and J. Gu, "Similarity-based online feature selection in content-based image retrieval," *IEEE Transactions on Image Processing*, vol. 15, no. 3, pp. 702–712, 2006.

[71] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[72] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[73] J. G. Dy and C. E. Brodley, "Feature selection for unsupervised learning," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 845–889, 2004.

[74] M. H. Law, M. A. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[75] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 301–312, 2002.

[76] S. K. Pal, R. K. De, and J. Basak, "Unsupervised feature evaluation: A neuro-fuzzy approach," *IEEE Transactions on neural networks*, vol. 11, no. 2, pp. 366–376, 2000.

[77] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *NIPS*, vol. 186, 2005, p. 189.

[78] Y. Yang, H. T. Shen, Z. Ma, Z. Huang, and X. Zhou, "l2, 1-norm regularized discriminative feature selection for unsupervised learning."

[79] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, NY, USA: ACM, 2010, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1835804.1835848

[80] X. Wang, Y. Wang, and L. Wang, "Improving fuzzy c-means clustering based on feature-weight learning," *Pattern Recognition Letters*, vol. 25, no. 10, pp. 1123 – 1132, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865504000765

[81] R. Zhang and Z. Lu, "Large scale sparse clustering."

[82] J. G. Dy and C. E. Brodley, "Feature subset selection and order identification for unsupervised learning," in *ICML*. Citeseer, 2000, pp. 247–254.

[83] M. Dash, H. Liu, and J. Yao, "Dimensionality reduction of unsupervised data," in *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, Nov 1997, pp. 532–539.

[84] S. Basu, C. A. Micchelli, and P. Olsen, "Maximum entropy and maximum likelihood criteria for feature selection from multivariate data," in *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, vol. 3, 2000, pp. 267–270 vol.3.

[85] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 1, pp. 1:1–1:58, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1497577.1497578

[86] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 90–105, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1007730.1007731

[87] E. Müller, S. Günnemann, I. Assent, and T. Seidl, "Evaluating clustering in subspace projections of high dimensional data," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1270–1281, 2009.

[88] C.-H. Cheng, A. W. Fu, and Y. Zhang, "Entropy-based subspace clustering for mining numerical data," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 84–93. [Online]. Available: http://doi.acm.org/10.1145/312129.312199

[89] K. Kailing, H.-P. Kriegel, and P. Kröger, "Density-connected subspace clustering for high-dimensional data," in *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 2004, pp. 246–256.

[90] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[91] H.-P. Kriegel, P. Kroger, M. Renz, and S. Wurst, "A generic framework for efficient subspace clustering of high-dimensional data," in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.

[92] I. Assent, R. Krieger, E. Müller, and T. Seidl, "Inscy: Indexing subspace clusters with in-process-removal of redundancy," in *2008 Eighth IEEE International Conference on Data Mining*, Dec 2008, pp. 719–724.

[93] J.-W. Chang and D.-S. Jin, "A new cell-based clustering method for large, high-dimensional data in data mining applications," in *Proceedings of the 2002 ACM Symposium on Applied Computing*, ser. SAC '02. New York, NY, USA: ACM, 2002, pp. 503–507. [Online]. Available: http://doi.acm.org/10.1145/508791.508886

[94] B. Liu, Y. Xia, and P. S. Yu, "Clustering through decision tree construction," in *Proceedings of the Ninth International Conference on Information and Knowledge Management*, ser. CIKM '00. New York, NY, USA: ACM, 2000, pp. 20–29. [Online]. Available: http://doi.acm.org/10.1145/354756.354775

[95] C. M. Procopiuc, M. Jonesý, P. K. Agarwal, and T. M. Muraliý, "A monte carlo algorithm for fast projective clustering." ACM Press, 2002, pp. 418–427.

[96] Y. Li, M. Dong, and J. Hua, "Localized feature selection for clustering," *Pattern Recognition Letters*, vol. 29, no. 1, pp. 10–18, 2008.

[97] Y. Kim, W. N. Street, and F. Menczer, "Feature selection in unsupervised learning via evolutionary search," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 365–369.

[98] J. H. Friedman and J. J. Meulman, "Clustering objects on subsets of attributes (with discussion)," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 66, no. 4, pp. 815–849, 2004. [Online]. Available: http://dx.doi.org/10.1111/j.1467-9868.2004.02059.x

[99] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research*, vol. 5, no. Oct, pp. 1205–1224, 2004.

[100] K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee, "Findit: a fast and intelligent subspace clustering algorithm using dimension voting," *Information and Software Technology*, vol. 46, no. 4, pp. 255–271, 2004.

[101] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[102] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1509–1517. [Online]. Available: http://papers.nips.cc/paper/3749-locality-sensitive-binary-codes-from-shift-invariant-kernels.pdf

[103] A. Broder, "On the resemblance and containment of documents," in *Proceedings of the Compression and Complexity of Sequences 1997*, ser. SEQUENCES '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 21–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829502.830043

[104] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian, "Super-bit locality-sensitive hashing," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 108–116. [Online]. Available: http://papers.nips.cc/paper/4847-super-bit-locality-sensitive-hashing.pdf

[105] O. Chum, M. Perd'och, and J. Matas, "Geometric min-hashing: Finding a (thick) needle in a haystack," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 17–24.

[106] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 1, pp. 66–78, 2012.

[107] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 512–518. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009730

[108] H. Xu and R. N. J. Veldhuis, "Binary representations of fingerprint spectral minutiae features," in *2010 20th International Conference on Pattern Recognition*, Aug 2010, pp. 1212–1216.

[109] H. Xu, R. N. J. Veldhuis, T. A. M. Kevenaar, and T. A. H. M. Akkermans, "A fast minutiae-based fingerprint recognition system," *IEEE Systems Journal*, vol. 3, no. 4, pp. 418–427, Dec 2009.

[110] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 817–824.

[111] K. Grauman and R. Fergus, "Learning binary hash codes for large-scale image search," in *Machine learning for computer vision*. Springer, 2013, pp. 49–87.

[112] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 604–613. [Online]. Available: http://doi.acm.org/10.1145/276698.276876

[113] M. M. Esmaeili, R. K. Ward, and M. Fatourechi, "A fast approximate nearest neighbor search algorithm in the hamming space," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 12, pp. 2481–2488, 2012.

[114] S. Brin, "Near neighbor search in large metric spaces," in *Proceedings of the 21th International Conference on Very Large Data Bases*, ser. VLDB '95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 574–584. [Online]. Available: http://dl.acm.org/citation.cfm?id=645921.673006

[115] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Computer and Robot Vision (CRV), 2012 Ninth Conference on*. IEEE, 2012, pp. 404–410.

[116] M. Muja and D. Lowe, "Flann-fast library for approximate nearest neighbors user manual," *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, 2009.

[117] K.-I. Lin, H. V. Jagadish, and C. Faloutsos, "The tv-tree: An index structure for high-dimensional data," *The VLDB Journal*, vol. 3, no. 4, pp. 517–542, Oct 1994. [Online]. Available: https://doi.org/10.1007/BF01231606

[118] C. Yu, B. C. Ooi, K. lee Tan, and H. V. Jagadish, "Indexing the distance: An efficient method to knn processing," 2001.

[119] F. Camastra, "Data dimensionality estimation methods: a survey," *Pattern recognition*, vol. 36, no. 12, pp. 2945–2954, 2003.

[120] F. Camastra and A. Vinciarelli, "Estimating the intrinsic dimension of data with a fractal-based method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 10, pp. 1404–1407, Oct 2002.

[121] C. Faloutsos and I. Kamel, "Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension," in *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 1994, pp. 4–13.

[122] K. W. Pettis, T. A. Bailey, A. K. Jain, and R. C. Dubes, "An intrinsic dimensionality estimator from near-neighbor information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 1, pp. 25–37, Jan 1979.

[123] T. Martinetz and K. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, no. 3, pp. 507–522, 1994.

[124] M. Kirby, *Geometric data analysis: an empirical approach to dimensionality reduction and the study of patterns*. John Wiley & Sons, Inc., 2000.

[125] I. Jolliffe, "Principal component analysis," *Springer Series in Statistics, Berlin: Springer, 1986*, 1986.

[126] J. Karhunen and J. Joutsensalo, "Representation and separation of signals using nonlinear pca type learning," *Neural networks*, vol. 7, no. 1, pp. 113–127, 1994.

[127] A. Rozza, G. Lombardi, C. Ceruti, E. Casiraghi, and P. Campadelli, "Novel high intrinsic dimensionality estimators," *Machine learning*, vol. 89, no. 1-2, pp. 37–65, 2012.

[128] D. R. Karger and M. Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 741–750.

[129] M. E. Houle, H. Kashima, and M. Nett, "Generalized expansion dimension," in *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 587–594.

[130] M. E. Houle, "Local intrinsic dimensionality I: an extreme-value-theoretic foundation for similarity applications," in *International Conference on Similarity Search and Applications*. Springer, 2017, pp. 64–79.

[131] X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, M. E. Houle, G. Schoenebeck, D. Song, and J. Bailey, "Characterizing adversarial subspaces using local intrinsic dimensionality," *arXiv preprint arXiv:1801.02613*, 2018.

[132] M. E. Houle, "Inlierness, outlierness, hubness and discriminability: an extreme-value-theoretic foundation," 2015.

[133] S. Coles, J. Bawa, L. Trenner, and P. Dorazio, *An introduction to statistical modeling of extreme values*. Springer, 2001, vol. 208.

[134] B. M. Hill, "A simple general approach to inference about the tail of a distribution," *Annals of Statistics*, vol. 3, no. 5, pp. 1163–1174, 1975.

[135] J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders, "The Amsterdam Library of Object Images," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103–112, 2005.

[136] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[137] M. E. Houle, V. Oria, S. Satoh, and J. Sun, "Knowledge propagation in large image databases using neighborhood information," in *ACM MM*, 2011, pp. 1033–1036.

[138] M. Lichman, "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[139] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *ESANN*, 2013.

[140] W. Ugulino, D. Cardador, K. Vega, E. Velloso, R. Milidiú, and H. Fuks, "Wearable computing: Accelerometers' data classification of body postures and movements," in *Advances in Artificial Intelligence-SBIA 2012*. Springer, 2012, pp. 52–61.

[141] K. Fernandes, P. Vinagre, and P. Cortez, "A proactive intelligent decision support system for predicting the popularity of online news," in *Portuguese Conference on Artificial Intelligence*. Springer, 2015, pp. 535–546.

[142] R. T. Ng and J. Han, "Clarans: a method for clustering objects for spatial data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1003–1016, Sep 2002.

[143] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[144] C. Bauckhage, "Numpy / scipy recipes for data science: k-medoids clustering," University of Bonn, Bonn, Nordrhein-Westfalen, Germany, Technical Report, Tech. Rep., 2015.

[145] W. Ugulino, D. Cardador, K. Vega, E. Velloso, R. Milidiú, and H. Fuks, "Wearable computing: Accelerometers' data classification of body postures and movements," *Advances in Artificial Intelligence-SBIA 2012*, pp. 52–61, 2012.

[146] K. Fernandes, P. Vinagre, and P. Cortez, "A proactive intelligent decision support system for predicting the popularity of online news," in *Portuguese Conference on Artificial Intelligence*. Springer, 2015, pp. 535–546.

[147] R. Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jiřina, J. Klaschka, E. Kotrč, P. Savický, S. Towers, A. Vaiciulis, and W. Wittek, "Methods for multidimensional event classification: a case study using images from a cherenkov gamma-ray telescope," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 516, no. 2, pp. 511 – 528, 2004.

[148] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek, "A framework for clustering uncertain data," *PVLDB*, vol. 8, no. 12, pp. 1976–1979, 2015. [Online]. Available: http://www.vldb.org/pvldb/vol8/p1976-schubert.pdf

[149] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[150] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, Dec 1985. [Online]. Available: https://doi.org/10.1007/BF01908075

[151] M. E. Houle, "The relevant-set correlation model for data clustering," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 1, no. 3, pp. 157–176, 2008.

[152] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," 1999.

[153] M. E. Houle, V. Oria, and A. M. Wali, "Improving k-nn graph accuracy using local intrinsic dimensionality," in *International Conference on Similarity Search and Applications*. Springer, 2017, pp. 110–124.

[154] J. Handl, J. Knowles, and D. B. Kell, "Computational cluster validation in post-genomic data analysis," *Bioinformatics*, vol. 21, no. 15, pp. 3201–3212, 2005.

[155] G. Brock, V. Pihur, S. Datta, S. Datta *et al.*, "clvalid, an r package for cluster validation," *Journal of Statistical Software (Brock et al., March 2008)*, 2011.

[156] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, "Fast algorithms for projected clustering," in *ACM SIGMoD Record*, vol. 28, no. 2. ACM, 1999, pp. 61–72.

[157] M. L. Yiu and N. Mamoulis, "Frequent-pattern based iterative projected clustering," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE, 2003, pp. 689–692.