An Approximate Nearest Neighbor Retrieval Scheme for Computationally Intensive Distance Measures

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science

by

Pratyush Bhatt
200402032
bhatt@research.iiit.ac.in

CVIT
International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2010

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "An Approximate Nearest Neighbor Retrieval Scheme for Computationally Intensive Distance Measures" by Mr. Pratyush Bhatt, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Anoop M. Namboodiri

To Family and Friends

Acknowledgments

This work would not have been possible without the help and support of many individuals. I offer my sincerest gratitude to my advisor, Dr. Anoop Namboodiri, who has supported me throughout my thesis with his patience and knowledge. I thank him for the freedom, he gave me to pursue my topic of interest. The trust, he showed on me by never forcing me to attend regular meetings, motivated me and developed my confidence to work on a problem independently.

I want like to thank my CVIT labmates and seniors, Avinash Sharma, Gopal Dutt Joshi and Naveen Tewari for their guidance and support during all these years. I would also like to thank my friends, Anshul, Chhaya, Gaurav, Maneesh, Prachi, Saurav, Saini, Sheetal, Sumit and Wasif for their company and making my stay at college memorable and entertaining. The many other people of note include Abheet, Anuj, Arjun, Batra, Dinkar, Gopal, Kochar, Mohit, Shishir, Shukla and Singla.

Abstract

Nearest neighbor retrieval can be defined as the task of finding the objects that are most similar to a query from a given a database of objects. It find its application in areas ranging from medical domain, financial sector, computer vision, computational sciences, computational geometry, information retrieval, etc. With the expansion of internet, the amount of digitized data is increasing by leaps and bounds. Retrieval of nearest nearest neighbors accurately and efficiently becomes challenging in such a scenario as the database contain a large number of objects. The problem gets worsen when the underlying distance measure used to compute [dis]similarity is computationally expensive. In such a scenario, sequential scan of data would take a lot of time which is the biggest problem for any online retrieval system. For example in biometric authentication systems, a particular person's biometric template is compared against all the registered samples in a database to identify the person. This process can be extremely time consuming in large databases even if the matching algorithm is extremely fast. For example, to do background check of a person who is crossing the border using the complete IAFIS,(a biometric person identification system at the U.S. border crossings), requires around 55 million comparisons. Even with the state of the art matching algorithms and computing facilities, this would take close to 10 minutes, which is not practical considering the millions of people who cross the border every month. Even for criminal investigations, it is desirable to get a quick and approximate search done immediately rather than the typical turn-around time of a few days for a search.

This thesis proposes a novel method for improving the efficiency and accuracy of nearest neighbor retrieval and classification in spaces with computationally expensive distance measures. The proposed technique is domain-independent, and can be applied in arbitrary spaces, including non- Euclidean and non-metric spaces. The main contributions of our work are :

- A representation scheme for objects in a dataset that allows for fast retrieval of approximate nearest neighbors in non-euclidean space. The approach named Hierarchical Local Maps (HLM),make use of manifold learning techniques to compute linear approximation of local neighborhoods.

- Search mechanism combined with filter and refine approach is proposed that minimizes the number of exact distance computations for computationally expensive distance measure.

- Study performance of our scheme on biometric data and study the parameters affecting its performance.

Results of k-nearest neighbor retrieval as well as classification results on UNIPEN dataset shows the advantages of using HLM over state-of-the-art approximate nearest neighbor retrieval algorithms. Classification result on CASIA iris dataset by using average gabor response for a block as the feature vector along with Euclidean distance as the soft biometric measure in conjugation with Daugman's feature vector and hamming distance as the hard biometric shows the advantage of using a softer metric over a hard metric for indexing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Finding an object similar to a given example from a huge set of potential candidates is one of the most common problems in data processing. Here the object could be a document, image, audio clip, video, etc. With the amount of digital data increasing many folds every year, a searching scheme that would retrieve similar objects with high accuracy and efficiency, is becoming essential. It finds its application in a variety of fields ranging from medical domain, financial sector, computer vision, computational sciences, computational geometry, information retrieval, etc.

Usually an object is characterized by the qualities it possesses, which signifies its identity. For a particular set of objects, either all the objects would be represented using a fixed set of parameters characteristic to the set, or each object may have its own defined parameters. The former case leads to a fixed length representation of an object whereas the latter leads to a variable length representation. If the number of parameters of the object are fixed, i.e., it has a fixed length representation, then an object can be visualized as a point in a multi-dimensional space whose dimensionality is equal to the cardinality of the parameter set. For example, if a person is characterized by only his height and weight, then every person will be mapped to a unique point in a $2D$ space having weight as one of its axis and height as the other. In many scenarios, the number of distinct qualities present in an object would differ. For example, assume a set of bags containing colored balls. The number of balls present in the bag and property of each ball (radius/color) could be a representation for a bag. As number of color balls differ from bag to bag, it leads to a variable length representation scheme.

In fixed length representations, the concept of similarity, i.e. how similar is one person to another, can easily be mapped to the distance between the corresponding two points in the $2D$ space. A function that computes the (dis)similarity between two objects in either fixed or variable length representation is called the similarity function or a distance measure. Similarity function can be understood as a black-box that would take the two objects as input and return a value that would indicate the similarity between these two objects. Thus naturally it would work for both fixed as well as variable length representations of the object.

The set of objects similar to a given sample forms its neighborhood. Even within a neighborhood, a closer object would be more similar than a farther one. Nearest neighbor retrieval problem can now be

formalized as retrieving objects similar to a given object, where similarity is in accordance to a given similarity function. A naive method for searching for a near similar object in a large set of objects, would be to find the similarity of the given object with all those in the set, in accordance to a given similarity function. The time taken for retrieval, would naturally increase with the dataset size and the average time taken for comparing two objects using the similarity function. This would become prohibitively expensive in very large datasets, which are becoming common.

To tackle this problem, various indexing methods were proposed by researchers. The main objective of any indexing method is to arrange the objects in an order (sometimes using precomputed inter-object similarity values) such that nearest neighbor retrieval becomes faster. i.e., similar objects are retrieved on comparing the query object with only a few dataset objects. Efficiency of indexing schemes highly depend on the nature of the similarity function, the number of features used to represent an object (dimensionality) and the distribution of the objects in the space (for fixed length representation).

A novel method for improving the accuracy and efficiency of nearest neighbor retrieval and classification in spaces with computationally expensive distance measures has been proposed in this thesis. In this chapter, we introduce the concept of indexing for nearest neighbor retrieval and classification. We highlight the applicability of such methods in various streams along with the application areas where underlying distance measure is computationally expensive. A brief overview of the main contributions of the thesis is given at the end.

1.1 Nearest Neighbor Retrieval: Issues and Applications

In recent years, there has been a dramatic increase in the capacity of digital storage systems. Enormous amounts of web content in large databases is stored an processed by web search engines helping Internet users quickly identify content of interest.

Large databases of biological data is processed to obtain information about biological structures of different species [12]. In computer vision and pattern recognition, problems like face recognition [35], body pose estimation [43], optical character recognition [8] require processing on enormous amount of data. In many applications the main purpose of a database is to provide a particular information on a need-to-know basis depending on a specific task, which is not known to the database system in advance. For example, problem of retrieving web pages similar to a particular page, proteins that are similar to a particular protein, etc.

With amount of data increasing, it becomes challenging to quickly extract the information from the database as the specific data that we are interested in at a particular moment is typically a small fraction of the entire database. The larger the database, the deeper the few items of interest are buried inside the database, and the harder we have to look to identify those items.

Any retrieval system is evaluated on the basis of two key measures: accuracy and speed. To increase the accuracy of system, user must be provided with the most relevant results related to the query and should mimimise the inclusion of the unrelated objects. Also, this operation should have a low process-

ing time as the waiting time for the user should also be minimized. In cases, where computationally expensive distance measures are used to compute similarity/dissimilarity between two samples, one should attempt to reduce such computations for a given input query. For example in biometric authentication systems, a particular person's biometric sample is compared against all the registered samples in a database to identify the person. This process can be extremely time consuming in large databases even if the matching algorithm is extremely fast. For example, to do background check of a person who is crossing the border using the complete IAFIS (a biometric person identification system), one needs to do around 55 million comparisons. Even with the state of the art matching algorithms, this would take close to 10 minutes, which is not practical considering the millions of people who cross the border every month. Even for criminal investigations, it is desirable to get a quick and approximate search done immediately rather than the typical turn-around time of a few days for a search.

Apart from Web browsing, biological databases and face recognition, nearest neighbor retrieval finds its application in variety of other problems such as:

- Optimizing network usage in peer-to-peer computer networks. Using the concept of network proximity, network performance is optimized while downloading specific content [26].
- Content-based retrieval systems process large multimedia databases to identify and efficiently retrieve documents, images and videos similar to a particular query [50].
- In the medical domain, given a new case, it is beneficial to identify and study the most similar cases in a database of case histories, to evaluate different treatment options [13].
- Applications requiring analysis and prediction of time series data, like stock prices, weather and climate data [31].
- Visualizing high-dimensional or non-Euclidean data, needs embedding of data onto a low dimensional space This requires identifying the nearest neighbors of each object and preserving local similarity [48].

While the methods proposed in this thesis can be used in many of the above retrieval applications, the main motivation for developing these methods has been to improve the accuracy and efficiency of nearest neighbor classifiers that use computationally expensive distance measures. We will now proceed to take a closer look at the topic of nearest neighbor classification.

1.2 Nearest Neighbor Classification

Suppose we want to build an optical character recognition system that can recognize images of isolated digits, from 0 to 9, as shown in Figure 1.1. Nearest neighbor classifier can be applied in such a case. A database of training images is first created with the class label indicating the class to which it belongs i.e. from 0 to 9. A distance measure needs to be specified that would compute the similarity

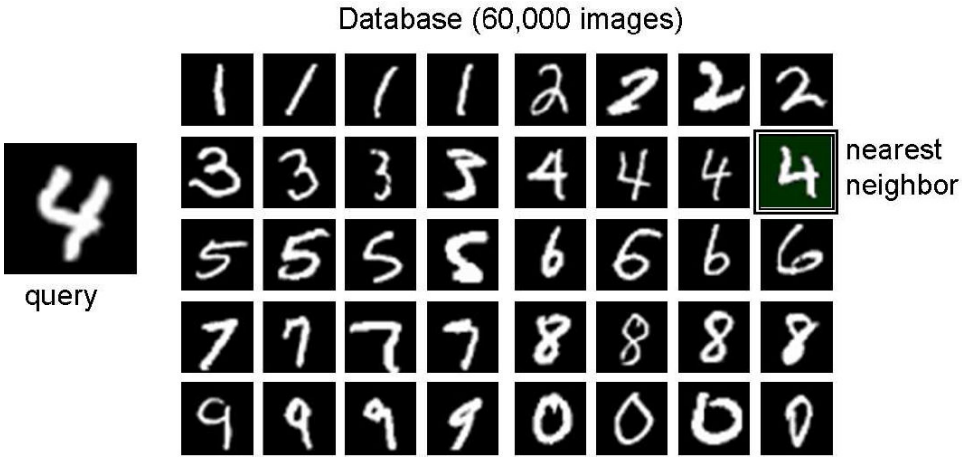


Figure 1.1: Handwritten digit recognition using a nearest neighbor classifier and the MNIST database of 60,000 training images [8]. Given a query image that we want to classify, the system retrieves the nearest neighbor of the query in the database, and assigns the class of the retrieved nearest neighbor to the query image.

between two images. Thus, when a query image is given, we compute the nearest neighbor of the query image and based on the class label of the nearest sample, the class for the query sample is assigned. Alternatively, k-nearest neighbor can be used to classify the test image.

Nearest neighbor classification is a popular technique in computer vision and pattern recognition. One of the main attractions of nearest neighbor classifiers is their simplicity. All we need to do to design such a classifier is provide a database of training objects and specify a distance measure. At the same time, nearest neighbor classifiers can be very powerful and have some very desirable properties:

- Nearest neighbor classifiers are easily scalable to large number of classes. Many popular methods like AdaBoost [42] and support vector machines [47], are not well-suited for such problems because they do not scale well with the number of classes.
- It is applicable for small class problem, such as optical character recognition, because of their ability to model complex, non-parametric distributions.
- It has been proved that the accuracy of k-nearest neighbor classifiers becomes asymptotically optimal as the training size approaches infinity [18].

In practice, nearest neighbor classifiers are often more accurate than other, significantly more complicated classification methods. As an example, nearest neighbor classification using shape context matching as a distance measure produced a lower error rate than a large number of competing methods for the problem of handwritten digit recognition, as measured on the popular MNIST dataset [8].

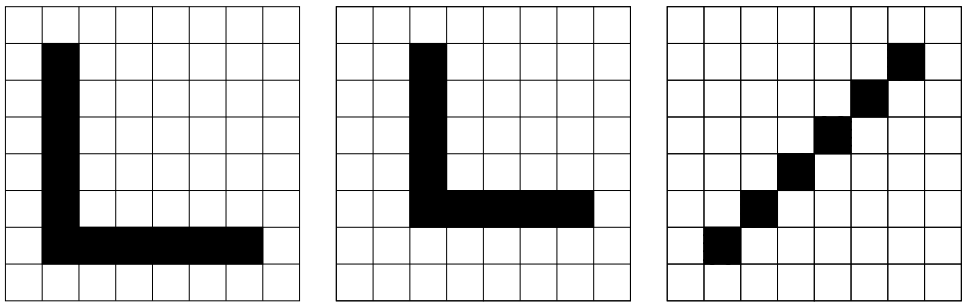
However, computationally expensive distance functions make applicability of nearest neighbor classifiers often impractical for real applications. The handwritten digit recognition system in [8] takes over

20 minutes to classify a single object on a modern PC using an optimized C++ implementation. Larger the available training data, better will be the accuracy but at the cost of high computation time.

The main focus of this thesis is to improve nearest neighbor retrieval and classification performance in spaces with computationally expensive distance measures. In the next section we provide some additional motivation for our method, by illustrating several examples where expensive distance measures are used to define notions of similarity. We also discuss the problems that arise when using such measures.

1.3 Computationally Expensive Distance Measures

To determine whether a distance measure is computationally expensive or not is highly subjective and application dependent. We consider a distance measure to be computationally expensive if computing distance/similarity between two objects take superlinear time to the length of these objects. Time complexity for computing L_p distance between two objects is of order $O(d)$, where d denotes dimensionality of the object representation. Most common examples of L_p metric are L_2, i.e. Euclidean distance and L_1, which is Manhattan distance Though computing this metric too in high dimensions takes lot of time but still we consider them efficient alternative to the computationally expensive other distance measure that are applied to the objects. Consider the problem of optical character recognition



X_1 X_2 X_3

Figure 1.2: Need for computationally expensive distance measure. Though X_1 is visually more similar to X_2 than X_3 but L_1 distance between X_1 and X_2 is more than the L_1 distance between X_1 and X_3.

seen in Figure 1.1. In such a problem sometimes we need to deal with the space of binary edge images. If we represent the edge image with a binary vector, with one dimension per image pixel. A value of 1 in a dimension represents it is a edge pixel. Now as Figure 1.2 shows, computing manhattan distance between two images discards the inherent nature of the image and though X_1 is more similar to X_2 in structure, its L_1 distance to X_3 is less than then distance to X_2. In such a space it is more beneficial to use a computationally expensive distance measure like chamfer distance [6] and the Hausdorf distance [28] as seen in Figure 1.3. The directed chamfer distance between two edge image A , B is the average distance from each edge pixel in A to its nearest edge pixel in B. The undirected chamfer distance,

which is often referred to simply as chamfer distance, is the sum of the two directed distances, from A to B and from B to A. The Hausdorf distance is the maximum distance between an edge pixel in one image and its nearest edge pixel in the other image. Figure 1.3 shows computation of these distances. For the images shown in Figure 1.2, the chamfer distances between X_1 and X_2 is 2.04, between X_1 and

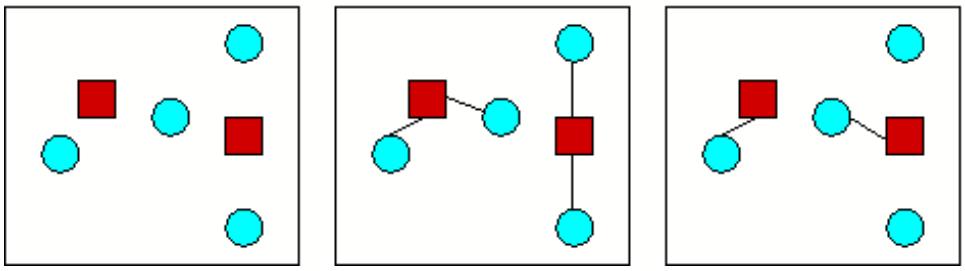


Figure 1.3: An example of the chamfer distance and the hausdorf distance. The left image shows the two set of points, square denotes one set of point whereas circle denotes another set. The middle image shows the link between the circle and its corresponding closest square. Average length of these links denotes the directed chamfer distance between set of circles and set of squares. The right image shows the link between the square and its corresponding closest circle. The directed chamfer distance between set of squares and set of circles is the average length of these link. Now the undirected chamfer distance between two sets is the sum of these two directed distances. The length of the longest link in middle and right image is the hausdorf distance between two set of points.

X_3 is 4.52, and 3.56 between X_2 and X_3, using the Euclidean distance to compute distances between pixel locations. Time complexity for both the chamfer distance and the Hausdorf distance is $O(dlogd)$, for atmost d edge pixels [28]. If for each edge image, distance transform is precomputed, then it takes $O(d)$ time to compute the distance [14]. For a large database, storing distance transforms of edge images can increase the memory and disk storage requirements of that database by orders of magnitude, and thus using distance transforms is often impractical.

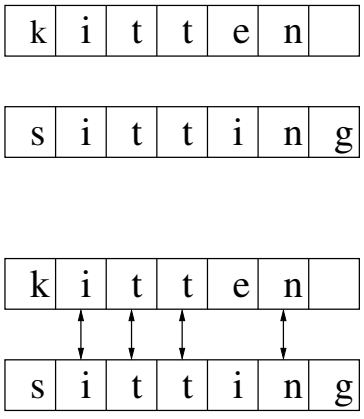


Figure 1.4: Figure shows two letters sitting and kitten; Number of letters matched between them in order are 4, thus DTW distance between them is 7-4=3.

Edit distance [33], which counts the minimum number of insertions, deletions and substitutions that are required to convert one string into the other string. Figure 1.4 shows the distance between two strings: *sitting* and *kitten*. It is another example of computationally expensive distance measure as it takes $O(d^2)$ time using dynamic programming. UNIX/LINUX diff utility uses edit distance to find the minimal set of changes that can be applied to convert one file to another.

Additional examples of computationally expensive distance measures are shape context matching [8] for comparing edge images, the Kullback-Leibler (KL) distance [15] for comparing distributions, Dynamic Space Time Warping [3] for comparing dynamic gestures and the Earth Mover's Distance (EMD) [41] to compute geodesic distance.

Use of such computationally expensive distance measures in a large database makes nearest neighbor retrieval challenging. In domains where the distance measure is computationally expensive, significant computational savings can be obtained by constructing a distance-approximating embedding, which maps objects into another space with a more efficient distance measure. In next section, we briefly look at the index methods, which is the first step for efficient retrieval in any space(metric/non-metric).

1.4 Indexing in Metric Space

The distance function D of a metric space (U, D) satisfies the following conditions,
$\forall x, y, z \in U$:

- **Non negativity**: $\forall s, t \in \mathbb{U}: \quad d(s,t) \geq 0$
- **Symmetry**: $\forall s, t \in \mathbb{U}: \quad d(s,t) = d(t,s)$
- **Identity:** $d(s,t) = 0 \Rightarrow s = t$
- **Triangle inequality:** $\forall r, s, t \in \mathbb{U}: \quad d(r,t) \leq d(r,s) + d(s,t)$

The properties of metric spaces allow some basic observations that can yield significantly faster algorithms for nearest-neighbor searching. These follow from the triangle inequality, which allows bounds on a distance we may not have computed, say $D(q, s)$, to be derived from two distances we may already know, say $D(q, p)$ and $D(p, s)$.

The bounds from the triangle inequality give a way to avoid computing the distance from a query point q to many of the sites, by giving bounds on their distance that allow the sites to be ruled out as nearest.

Various tree structures [20, 22] are applied by researchers to index multidimensional point. Though in high dimensions, owing to curse of dimensionality [7] many of the indexing techniques fail. Telescopic-Vector tree (TV Tree) [34], X- Tree [10] and Pyramid Tree [9] deal with this problem of dimensionality curse in high dimensions.

Other methods include dimensionality reduction techniques like PCA [30] , Factor Analysis [25] and Independent Component Analysis(ICA) [29]. The aim of these approaches is to reduce the dimensionality of the data and then use any metric tree structure or hashing based structure in the resulting low dimension.

1.5 Indexing in Non-Metric Space

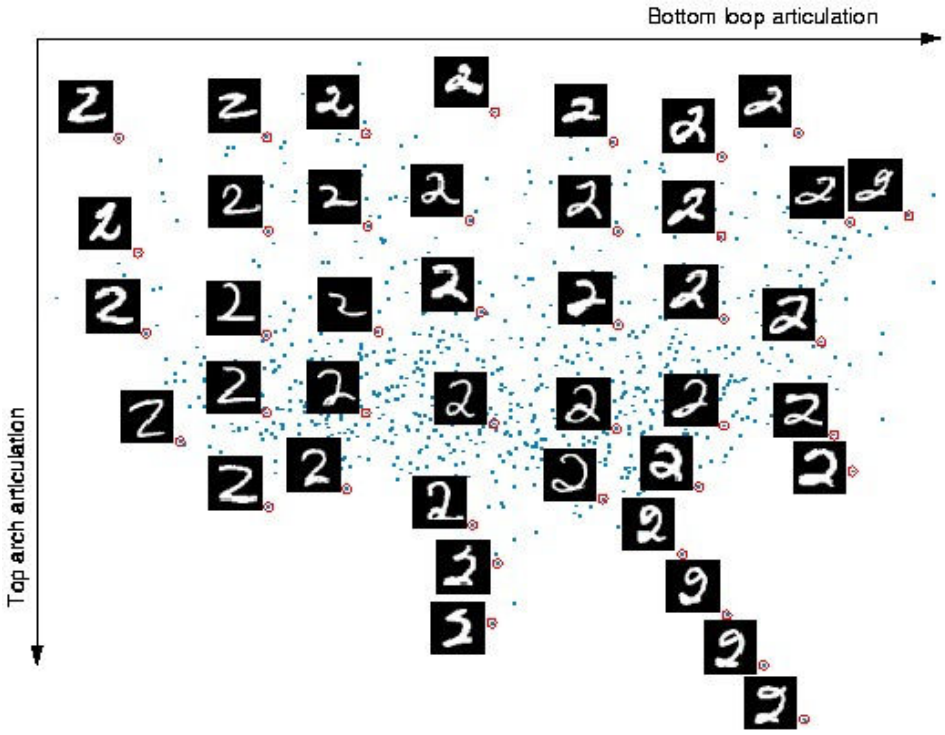


Figure 1.5: Embedding of the images of letter 2 onto a two dimensional space [45]. Bottom loop articulation increases as we move along the X-axis, whereas Y-axis denotes the extent of top arch articulation. If it is possible to project all the images onto this plane bases on these two parameters, then search for near similar digit would be easier in this euclidean space where any metric indexing schemes can be applied.

Consider the database of images containing image of letter 2 in Figure 1.5. If we consider one dimension for each pixel of the image, then it would be a high dimensional vector but that would distort the structural property of the letter. Consider the problem of retrieving closest neighbor for a given query image. Solving this problem in original space is too difficult as that would require a highly complex and computationally expensive distance function. If we could map all the images of the database on the basis of two properties namely: Top arch articulation and Bottom loop articulation, we could easily find the nearest neighbor in this $2D$ space, using L_2 distance. As seen in Figure 1.5, the bottom loop

articulation increases along X-axis and Top arch articulation increases along Y-axis. The only problem in such a approach is to finding a mapping function from high dimensional space to a low dimensional space that would scatter the data on the basis of hidden intrinsic dimensionality of the data.

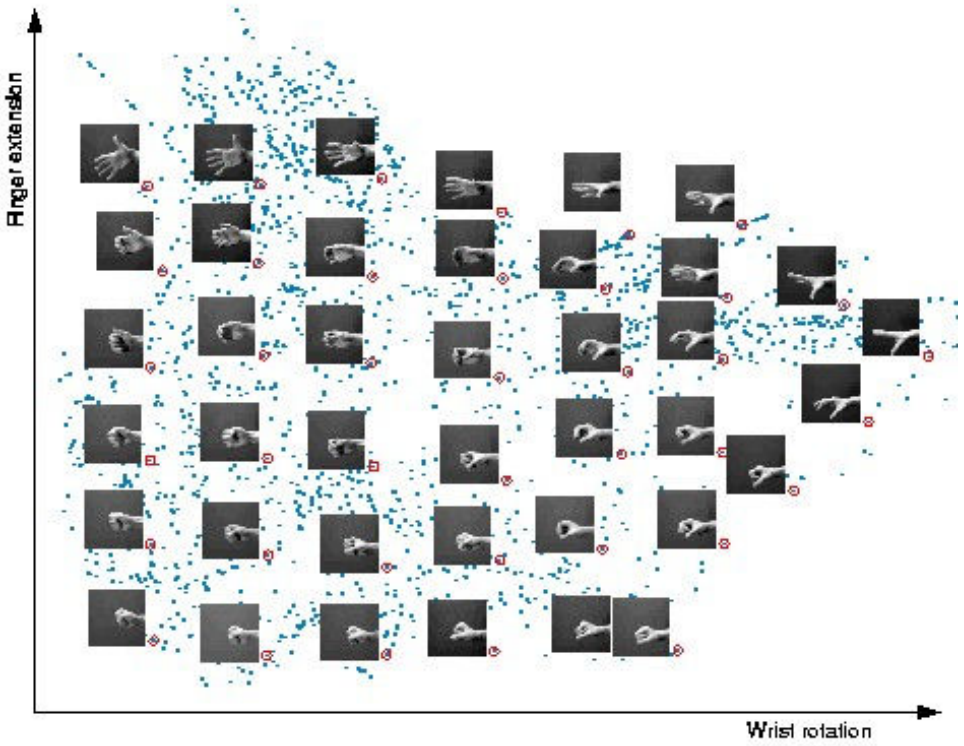


Figure 1.6: Embedding of the wrist images onto a two dimensional space [45]. Amount of wrist rotation increases as we move along the X-axis, whereas Y-axis denotes the extent of finger extension. Again searching would be easier, if it is possible to project all the images onto this plane bases on these two parameters.

Similarly in Figure 1.6, the hidden intrinsic dimensionality of data is only two. Each wrist image can be identified on the basis of two parameters: finger extension and wrist rotation. The goal is to find this embedding, where on moving along X-axis would map to increase in wrist rotation and moving along Y-axis depicting the amount of finger expansion. Algorithms like ISOMAP [45] and Local Linear Embedding [40], can be used to find such a low dimensional structure. Such methods are targeted for visualization purposes and not for fast retrieval of nearest neighbors. Methods like Bourgain embeddings [27] , FastMap [19] , Lipschitz Embedding [27], BoostMap [4] find low dimensional embedding for the data, where retrieval is made faster using filter and refine approach

1.6 Contribution

The most straightforward solution for nearest neighbor retrieval is brute-force search. The distance between the query object and each database object is computed. The time complexity of brute-force search is linear to two quantities: the number of database objects, and the average time it takes to measure the distance between two objects. Thus, with the increase in the number of database objects, brute-force search can become computationally demanding, or even impractical for particular applications. This problem is worsened in domains where evaluating the distance between two objects is computationally expensive. Further these superlinear computationally expensive distance measure are not L_p metric, and so many indexing techniques are not applicable in such a space. While several metric based methods are proposed to perform indexing in this space but many popular distance metric violates triangular inequality and thus are non-metric. Examples of non-metric measures are the chamfer distance [6], the Kullback-Leibler (KL) distance [15], Dynamic Space Time Warping [3] etc.

The main contributions of our work are :

- A representation scheme for objects in a dataset that allows for fast retrieval of approximate nearest neighbors in non-euclidean space.
- Search mechanism combined with filter and refine approach is proposed that minimizes the number of exact distance computations for computationally expensive distance measure.
- Study performance of our scheme on biometric data and study the parameters effecting its performance.

Chapter 2

Related Work

In this chapter, we provide a more detailed view of the various indexing schemes for efficient nearest neighbor retrieval in Euclidean and non-Euclidean spaces. Section 2.1 gives an insight to the existing methods to index multidimensional points in a metric space. Section 2.2 discusses the different schemes applied to index very high dimensional points in metric space. Section 2.3 studies approaches when input is pairwise dissimilarity rather then the actual points in multidimensional metric space. Section 2.4 reviews existing methods for efficient indexing when the underlying distance measure is computationally expensive. It also describes a popular retrieval framework, the filter-and-refine retrieval framework,that these embedding methods are typically used in conjunction with. Section 2.5 summarizes the various indexing techniques and their applicability, and highlights the properties of our method in comparison to these work.

2.1 Indexing Multidimensional Point Data

To index a multidimensional point data, the main emphasis is given on choosing an appropriate representation/ data structure that would facilitate operations such as search. Such multidimensional data exist in diverse fields including database management systems, computer graphics, game programming, geographic information systems, pattern recognition, similarity searching, computational geometry and numerous others. In such a representation, the data is sorted in a way that would make it more accessible while querying. To this end various approaches have been employed, that could be broadly categorized in two; namely tree based indexing [50] and hashing structures [52].

2.1.1 Tree Based Methods

The main emphasis of such methods is to do space partitioning. Space partitioning is the process of dividing a Euclidean space into multiple disjoint subsets (depending on the branching factor of the tree).

In other words, the input space is divided into non-overlapping regions. A point in such a space can lie in exactly one of the regions. Most of the Space-partitioning systems are hierarchical, i.e. a region of

space is divided into several regions, and then the same space-partitioning system is recursively applied to each of the regions thus created. Space-partitioning trees organize these regions in a tree structure. In a Euclidean space, such trees are also called metric trees, as they exploit properties of metric spaces such as the triangle inequality for efficient retrieval of the data .

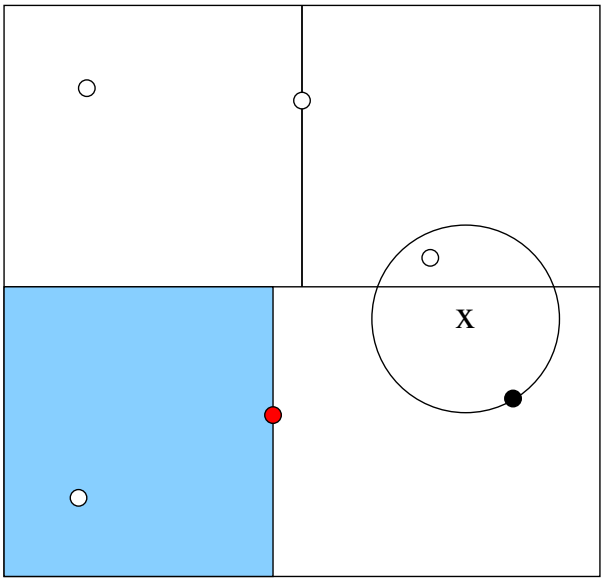


Figure 2.1: KD Tree Construction and Search .

Two of the most common metric trees are:

- KD-Tree : The optimized k-d tree [20] is probably the data structure most often used in practice for nearest neighbor searching in main memory. It is a form of Binary search tree, which is formed by a recursive sub-division of a d-dimensional input space by using a $d-1$ dimensional hyper-plane at each node. Figure 2.1 shows the process of KD-Tree construction. While performing search, a first approximation is initially found at the leaf node which contains the target point X. In Figure 2.1, the black dot is the dot which owns the leaf node containing the target X and is the first approximation of nearest neighbor of X. As it can be seen, the first approximation may not be the closest one, but the closest point must lie inside the circle with target point as center and distance between X and first approximation as the radius. The search now, backtrack to the parent node denoted by red dot in the Figure 2.1. We now explore the possibility of finding the closest point in parent's other child. As the circle does not intersect with the shaded region owned by parent's other child. If no closer neighbor can exist in the other child, the algorithm can immediately move up a further level and the same procedure is followed.

  Generally during a nearest neighbor search only a few leaf nodes need to be inspected, denoted by the white region in Figure 2.2. All the leaf nodes in shaded region need not be searched and thus search time is drastically reduced.

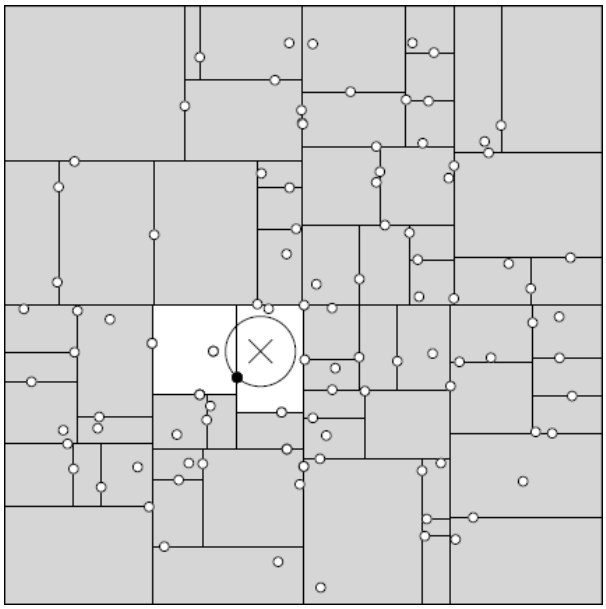


Figure 2.2: KD Tree Search [38].

- R-Tree [22] : The concept of bounding hyper-rectangles is applied, where each leaf node is a bounding rectangle that encloses all the child nodes it contains within it. It can be visualized as a b-tree by keeping each of its non-root nodes at least half-full resulting in a height balanced tree. The obvious problem with R-Tree is of having overlapping intermediate nodes, due to which multiple paths need to be searched during retrieval. Intuitively, overlap is the percentage of the volume that is covered by more than one hyper-rectangle. This intuitive definition of overlap,.is directly correlated to the query performance since in processing queries, overlap of directory nodes results in the necessity to follow multiple paths, even for point queries It can be seen from Figure 2.3, shaded region depicts the areas of overlap and black region denotes the area of multiple overlap. For points lying in this region, search would lead to multiple paths and thus the retrieval time increases. Experimentally it has been proved that with increasing dimensionality the problem of overlap only worsens.

2.1.2 Hashing Based Methods

In most of the database applications, a primary key is associated with each data which is used to index data. The main idea is to put similar/near similar objects in the same bin. The performance of the retrieval usually depend on the hash function applied to assign each point its corresponding bin as seen in Figure 2.4. Once a bin is identified with the given input query, linear traversal is used inside the bucket to find the most appropriate match.

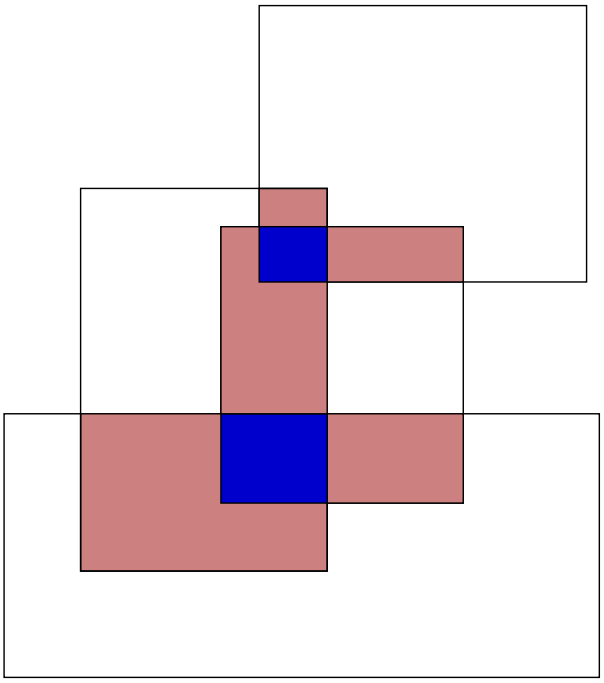


Figure 2.3: Overlap and Multi-overlap of 2 dimensional data.

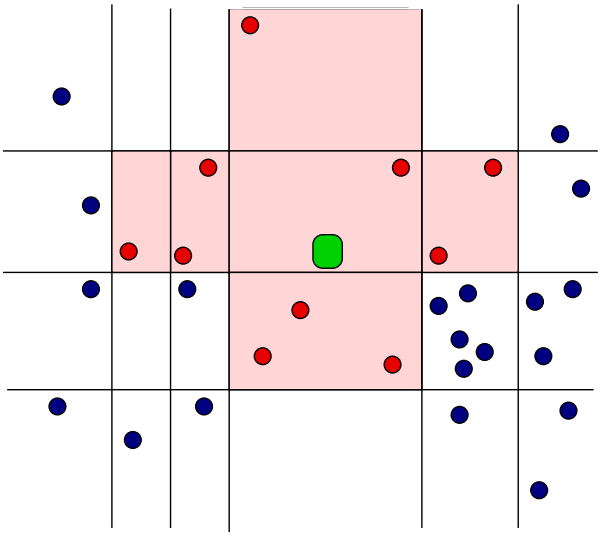


Figure 2.4: General Hashing based Indexing Scheme.

2.2 Indexing in High Dimensions

Search queries become increasingly difficult to solve as the dimensionality increases, due to *curse of dimensionality* [7], that states the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with the number of variables(i.e. , dimensions) that it comprises. In high dimensions, the points are at about the same distance between each other [11]. This

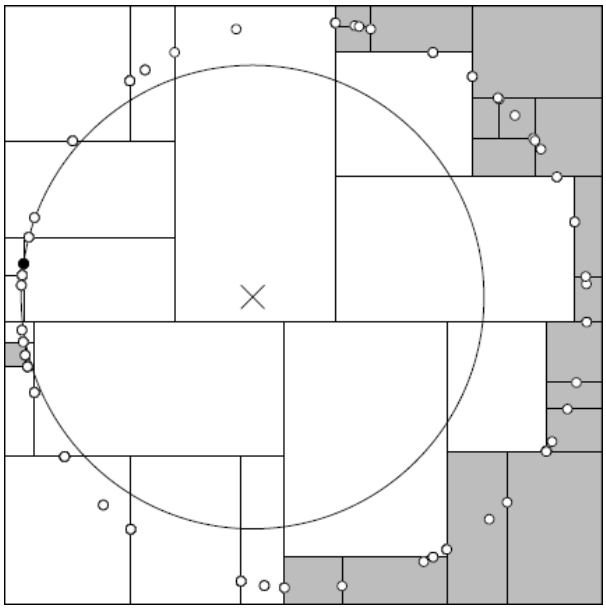


Figure 2.5: KD Tree Search in high dimensions [38]

distance grows with dimensionality and decrease marginally as the number of points increases [49]. The variance of inter-point pairwise distance approaches a constant value. As it can be seen in Figure 2.5, KD-tree visits almost all the leaf nodes of the tree, before retrieving the closest point. This phenomenon is formalized as:

$$lim_{D\to\infty}\frac{Distmax - Distmin}{Distmin} \to 0. \tag{2.1}$$

That is, the concept of similarity/dissimilarity between points no longer holds when the space dimensionality increases as all the points are equi-distant to each other.

This makes some of the concepts that we take for granted in low dimensional spaces meaningless in high dimensions. For example the concept of nearest neighbor is no more meaningful in such a scenario. Not only because all the points are almost at the same distance, but also because in such a configuration even a small perturbation can change the nearest point into the farthest one and vice-versa.

Space partitioning methods discussed in Section 2.2.1, becomes intractable as the space dimension grows. Dividing the space in half along each dimension generates 2^D partitions, each containing zero or a small number of points. Again the space is almost empty [49]. For example in a space of $D = 100$ the

first partition contains almost 10^{30} blocks. If the space contains 10^6 points, only one partition in 10^{24} contains a single point.

This is the reason for which in high dimensional spaces there are never enough data points. In the next section, we will see how researchers have handled the problem of indexing high dimensional data for nearest neighbor retrieval or approximate nearest neighbor retrieval.

2.2.1 Space Partitioning Trees

This subsection describes three major Space Partitioning trees modified to index high dimensional point.

- Telescopic-Vector tree (TV Tree) [34] : The main idea is to contract and extend the feature vectors dynamically, i.e., only few of the features that are necessary to discriminate among the objects are used. Intuitively we can see, even humans use this method to classify objects: for example, in zoology, the species are grouped in a few broad classes, using a few features (eg., vertebrates versus invertebrates). More and more features are gradually utilized for further classification (eg., the feature of warm-blood versus cold-blood, for the vertebrates; similarly, the feature of lungs versus branchia etc.). Compared to other tree structures, it provides a higher fanout at the top levels using only a few basic features, leaving the irrelevant ones. As more and more objects are inserted into the tree, more features might be needed for discrimination. The key point here is that features are introduced on a when needed basis and thus the 'dimensionality curse' is softened.

- Extended node Tree (X- Tree) [10] : The goal is to support not only point data but also extended spatial data and therefore, the X-tree uses the concept of overlapping regions. As it is clear that we have to avoid overlapping regions as shown in Figure 2.3 in order to improve the indexing of high-dimensional data, X-tree therefore avoids overlap whenever it is possible without allowing the tree to degenerate by introducing concept of extended variable size nodes, so-called supernodes. In addition to providing a directory organization which is suitable for high-dimensional data, the X-tree uses the available main memory more efficiently. The structure is insertion-order dependent.

- Pyramid Tree [9] : It is based on a special partitioning strategy that is optimized for high-dimensional data. The key idea is to divide the d dimensional data space first into $2d$ pyramids sharing the center point of the space as the top as seen in Figure 2.6. Subsequently, each of the single pyramid is cut into slices parallel to the base of the pyramid that forms the data pages. Such a partition strategy yields a mapping from the d-dimensional space to a 1-dimensional space. B+-tree is then applied to efficiently index these one dimensional transformed data.

2.2.2 Dimensionality Reduction Techniques

Another way to deal with high dimensional data is to transform the data in the high-dimensional space to a space of fewer dimensions, where various indexing techniques are applicable. This transfor-

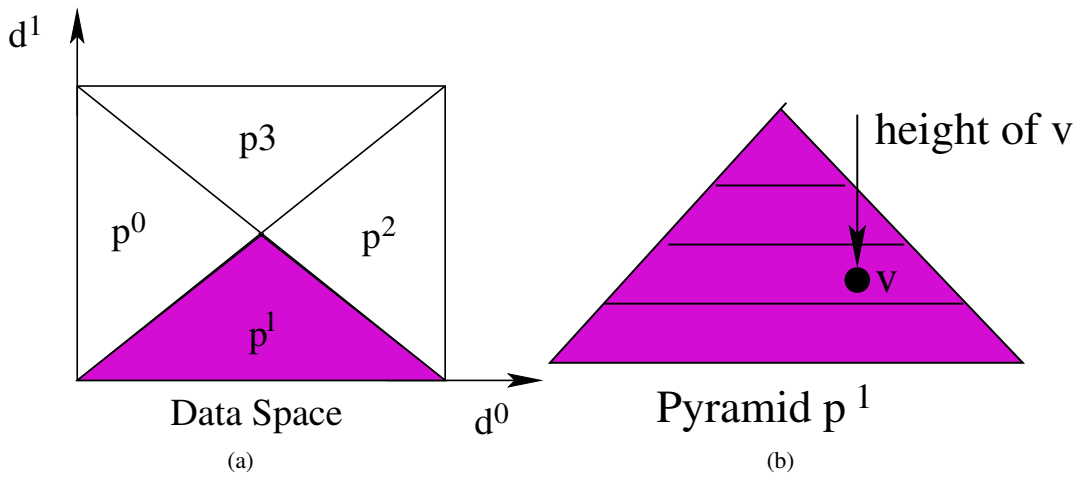


Figure 2.6: Indexing data using Pyramid Tree. a) $2D$ space is partitioned using 4 pyramids b) Height of point v in pyramid p_1 [9]

mation from high dimensional space to a low dimensional space can either be a linear transformation or a non-linear transformation.

Locality Sensitive Hashing (LSH) [39] is a method of performing probabilistic dimension reduction of high-dimensional data. The main idea is to hash the input items so that similar objects are mapped to the same buckets with high probability (the number of buckets being much smaller than the universe of possible input items).

The most popular linear dimensionality reduction approach is principal component analysis (PCA) [30], that performs a linear mapping of the high dimensional data to a lower dimensional space. The mapping is such that the variance of the data in the low-dimensional representation is maximized. The linear subspace is specified by d orthogonal vectors that form a new coordinate system, called the principal components. Only k principal components specified by corresponding top k eigen-values forms the new coordinate system as the first few eigenvectors can often be interpreted in terms of the large-scale physical behavior of the system. Factor Analysis [25] and Independent Component Analysis(ICA) [29] are few other approaches for linear dimensionality reduction

As visualization of high-dimensional data can be difficult to interpret one approach is to assume that the data of interest lies on an embedded non-linear manifold within the higher-dimensional space. Dimensionality reduction can also be seen as the process to find a set of degrees of freedom that reproduce most of the variability of a data set. Consider a set of images produced by the rotation of a face through different angles. Clearly only one degree of freedom is being altered, and thus the images lie along a continuous one-dimensional curve through image space. Figure 2.7 shows an example of image data that exhibits one intrinsic dimension.

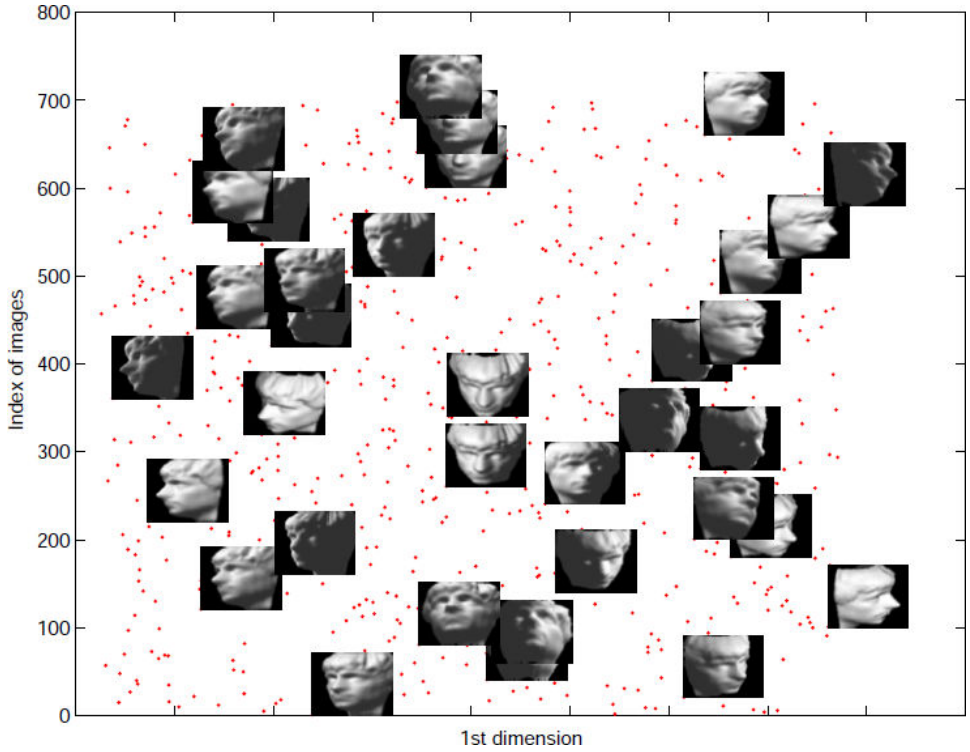


Figure 2.7: A canonical dimensionality reduction problem from visual perception. The input consists of a sequence of 4096-dimensional vectors, representing the brightness values of 64 pixel by 64 pixel images of a face. Applied to N = 698 raw images. The first coordinate axis of the embedding correlates highly with one of the degrees of freedom underlying the original data: left-right pose [45].

If the manifold is of low enough dimension then the data can be visualized in a low dimensional space. Various non-linear dimensionality reduction techniques like Kernel PCA [37], Local Linear Embedding(LLE) [40] and ISOMAP [45] have been proposed by the researchers.

- Kernel PCA : In Kernel PCA, through the use of kernels, principal components are efficiently computed in high-dimensional feature spaces that are related to the input space by some non-linear mapping. It finds principal components that are non-linearly related to the input space by executing PCA in the space produced by the nonlinear mapping, where the low-dimensional latent structure is, hopefully, easier to discover.

- Local Linear Embedding (LLE) : It computes a low-dimensional, neighborhood preserving embedding of the high-dimensional data. A set of local linear fits are used to recover the global nonlinear structure. It identifies the nonlinear structure of the data through two linear steps. First, the locally linear patches are computed and second, the linear mapping to the coordinate system on the manifold is done. The main goal here is to preserve relationship between neighboring points while mapping a high-dimensional data space to the single global coordinate system of the manifold.

- ISOMAP : It is a nonlinear generalization of Multidimensional Scaling (MDS) [51]. MDS is a classic dimensionality reduction approach where a low dimensional space that preserves pairwise distances between input points is computed. The main idea is to perform MDS, not in the input space, but in the geodesic space of the nonlinear data manifold. The geodesic distances represent the shortest paths along the curved surface of the manifold measured as if the surface were flat. It can be approximated by a sequence of short steps between neighboring sample points. Isomap then applies MDS to the geodesic rather than euclidean distances to find a low-dimensional mapping that preserves these pairwise distances.

The applicability of such methods are restricted to scenarios where data is lying on a manifold, i.e. there is a hidden intrinsic dimensionality of the data. Also it is assumed that metric properties hold for the data in its neighborhood. Methods discussed in this section like multidimensional scaling (MDS), locally linear embeddings (LLE) , and Isomap are not targeted at speeding up online similarity retrieval, because they still need to evaluate exact distances between the query and most or all database objects.

2.3 Distance Based Indexing

In many applications, easily identifiable features are not available. The only available information is a distance function that tells the degree of similarity(or dissimilarity) between a pair of object. These methods generally assume that a finite set S of N objects and distance metric d indicating the distance values between them is given. The distance based indexing schemas can be classified as:

- Ball Partitioning : The dataset is partitioned into two subsets based on distance from a single distinguished object, called vantage point depending on whether it is inside or outside a ball around the object. The asymmetry of ball partitioning is a potential drawback as the outer shell tends to be very narrow for metric space typically used in similarity searching. The vantage point tree(VP Tree) [21] is the most commonly used ball partitioning method. In this method, a pivot p from S is picked, and we compute the median r of the distance of the other objects t p, and then divide the remaining objects into two roughly equal-sized subsets S_1 and S_2 as follows:

$$S_1 = o \in S \setminus p | d(p, o) < r \tag{2.2}$$

$$S_1 = o \in S \setminus p | d(p, o) \geq r \tag{2.3}$$

  Applying this rule recursively leads to a binary tree, where a pivot object is stored in each internal node.

- Generalized Hyperplane Partitioning : Here instead of partitioning space based on a single distinguishable object, two distinguished objects p_1 and p_2 are chosen, and the dataset is partitioned

based on which of these two objects is the closest, i.e., the objects in subset A are closer to p_1 than to p_2, while the objects in subset B are closer to p_2. Unlike ball partitioning methods, it is more symmetric. Gh-Tree[46] is one such partitioning method. The rule that is recursively applied at each node of a binary is given by :

$$S_1 = o \in S \setminus p_1, p_2 | d(p, o) < r \tag{2.4}$$

$$S_1 = o \in S \setminus p_1, p_2 | d(p, o) \geq r \tag{2.5}$$

An alternate way of classifying the distance-based indexing methods is on the basis of whether they are pivot based of clustering based. The two approaches are explained as follows:

- Pivot Based : It selects a subset of the objects in dataset to serve as distinguished objects, termed *pivot objects* and classify the remaining objects in terms of their distances from the pivot objects. Known distance to different pivot objects are used to reduce the number of distance computations required by the query object to all the object in data set in a pivot based similarity search algorithm. AESA(Approximating and Eliminating Search Algorithm) [36] is a nearest neighbor algorithm that precomputes all pairwise interobject distances and stores it in a matrix. At query time, the distance matrix is used to provide a lower bound on the number of distances that need to be computed for a query sample. Triangular inequality is used to reduce the number of distance computations.

- Clustering Based : Here the complete data set is divided into spatial zones called clusters that are based on proximity to a distinguished object known as cluster center [44]. First, cluster centers are identified and then each point in data set is assigned the cluster to which its distance to cluster center is minimum. Generalized Hyperplane partitioning methods are examples of clustering based methods.

2.4 Embedding Methods for fast online retrieval

Many important applications require efficient nearest neighbor retrieval in non-Euclidean,and often non-metric spaces. In such a space, retrieving nearest neighbors efficiently can be challenging, as the underlying distance measures can take time superlinear to the length of the data.

Euclidean embeddings (like Bourgain embeddings [27] and FastMap [19]) provide an alternative for indexing non-Euclidean spaces. Using embeddings, each object is associated with a euclidean vector, so that distances between the vectors gets related to distances between objects. Input database objects are embedded offline.

For a given query object q, its embedding $F(q)$ is computed efficiently online, by calculating distance of q with a small number of database objects. The nearest neighbors of q, are found by first finding a

small set that contains the candidate natches using euclidean distance and then refining the result by measuring distance in original space.

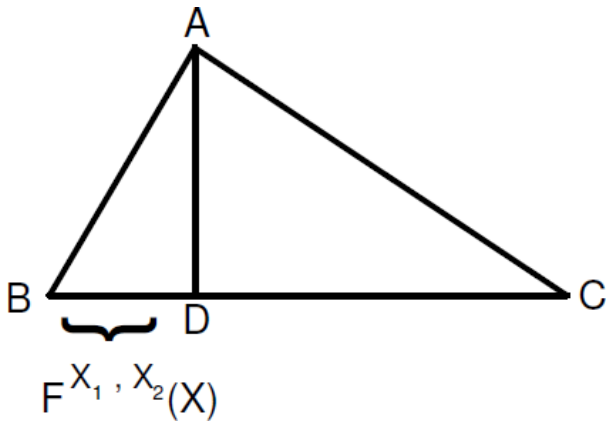


Figure 2.8: Fastmap embedding

Such euclidean embeddings can significantly improve retrieval time in domains where evaluating the distance measure in the original space is computationally expensive.

We first look at the various methods applied in order to embed input data in a space. Once the data is embedded into a low dimensional euclidean space, a *filter and refine* strategy [27] can be used to make online retrieval faster. The assumption is that distance measure D is computationally expensive and evaluating distances in Euclidean space is much faster. The filter step discards most database objects by comparing Euclidean vectors. The refine step applies D only to the top p candidates. This is much more efficient than brute-force retrieval, in which we compute distance between q and the entire database.

- FastMap : A set of simple $1D$ embedding [19] acts as a building block for FastMap.

  Two objects $x_1, x_2 \in X$, called pivot objects, are chosen and then, for an arbitrary $x \in X$, define the embedding as the projection of x onto the line x_1x_2. As seen in Figure 2.8, the projection can be defined by treating the distances between x, x_1, and x_2 as specifying the sides of a triangle in R^2:

  $$F^{x_1,x_2}(x) = \frac{D(x,x_1)^2 + D(x_1,x_2)^2 - D(x,x_2)^2}{2D(x_1,x_2)} \tag{2.6}$$

  Multiple pair of pivot objects are used to project the input space to a k dimensional space, using only $O(kn)$ computation of D.

  Although FastMap treats X as a Euclidean space, the resulting embeddings can be useful even when X is non-Euclidean, or even non-metric.

- Lipschitz Embedding [27] : Like FastMap, here too, the basic building block is the $1D$ embedding. A distinguished object often called pivot or reference object is selected from the database and all the points are projected on to a line based on the distance of their respective distance from

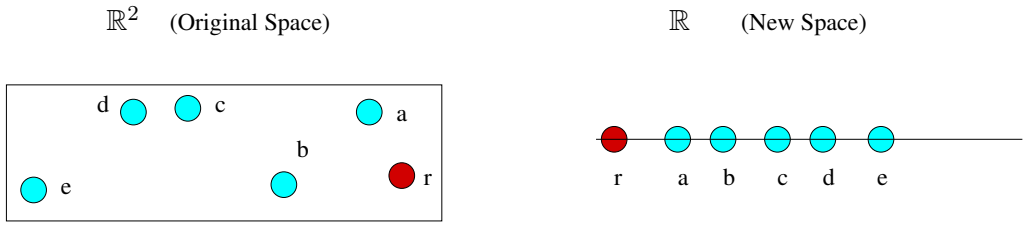


Figure 2.9: Lipschitz Embedding

the pivot. If the input space obeys triangular inequality, then the nearby points in input space will be mapped to nearby points in the $1D$ embedding. If the underlying distance function is non-metric, i.e. if triangular inequality is violated, nearby points in input space are mapped closer to each other most of the times. On the other hand, distant objects will map to nearby points if they have similar distance to reference object. To reduce the probability of distant objects falling to nearby point, a d dimensional embedding is formed by selecting d reference objects and computing distances to them. Mathematically, for any space X and distance measure D, we choose d reference set $P_1, P_2, ..., P_d$, the multidimensional embedding $F : X \rightarrow R^d$ is given by :

$$F(x) = (F^{P_1}(x), ..., F^{P_d}(x)) \tag{2.7}$$

Figure 2.9 shows embedding F^r of five $2D$ points onto a real line.

- BoostMap [4] : The key differentiating feature of BoostMap with respect to other embedding methods explained above is that it optimizes a direct measure of how well the embedding preserves the nearest neighbor structure of the original space, which is independent of the metric assumption and is valid in any space. Another difference being the machine learning formulation of the embedding approach used in BoostMap. The building block is again the $1D$ embedding that act as weak classifiers and given as input to AdaBoost [42], which combines multiple such weak classifiers into a strong classifier.

2.5 Summary

In short we can see that all the indexing schemes are data dependent. There is no single indexing approach that could work for all type of data and distance functions. If the data space is in low dimension and data is well populated along each dimension then tree based approaches or hashing methods as discussed in Section 2.1 can be applied to index the data.Unfortunately, most often that is not the case, as the input data is of high dimension where owing to curse of dimensionality such techniques fail. In such a case the main emphasis is on reducing the dimension of dataset and discarding irrelevant dimensions. Section 2.2.2 talks about the various dimensionality reduction techniques. These methods vary for data in linear and non-linear space. In non-linear case, the main goal is to learn a low dimensional embedding, where retrieval is faster as the resulting space would be Euclidean. Such embedding

techniques work for computationally expensive non-metric distance measures too. If instead of feature vectors, we have information about only the pairwise dissimilarity of input objects, then distance based indexing techniques discussed in Section 2.3 can be applied.

In this thesis, we are dealing with the computationally expensive distance measures. The main objective is to speed up the search by reducing the number of explicit distance measurements done using computationally expensive distance measure. Also, we are dealing with the scenario, where query is not known before hand, and thus retrieval should be done on the fly. Section 2.4 describes the work related to retrieval of nearest neighbor for classification of input query, that is most related to our work. Once the nearest neighbors are retrieved, a filter and refine approach is applied on top for classification purpose.

Chapter 3

Hierarchical Local Maps

In this chapter, we propose a novel method for fast nearest neighbors retrieval in non-Euclidean and non-metric spaces. We organize the data into a hierarchical fashion that preserves the local similarity structure. A method to find the approximate nearest neighbor of a query is proposed, that drastically reduces the total number of explicit distance measures that need to be computed. The representation overcomes the restrictive assumptions in traditional manifold mappings, while enabling fast nearest neighbor's search. We propose the use of local manifold mappings for finding robust and approximate k-nearest neighbors for a given sample. The method, referred to as Hierarchical Local Maps (HLM) (see Figure 3.1), arranges a set of simple local manifolds in a hierarchical fashion. As we move up in the hierarchy, the complexity of manifold increases as the data does not belong to a neighborhood. At the top most level, no lower dimensional space can be found, where the pairwise similarity is preserved. Nearest neighbor retrieval is now framed as selecting correct path to traverse down the hierarchy that would give approximate nearest neighbors. We present experiments on two real world complex dataset: the UNIPEN dataset [23], and the CASIA Iris dataset [1]. The results show a considerable amount of computationally expensive measurements can be reduced without affecting the accuracy of nearest neighbors found. We also present comparison to state of the art algorithms.

3.1 Hierarchical Local Maps

Goal: Given a set S of N points and an arbitrary distance measure (F) between them, construct a data structure that helps us to compute an approximate list of nearest neighbors of a query point q.

In most of the real-world datasets, there is no low-dimensional single manifold that spans the whole dataset. However, parts of the dataset may lie on a manifold. For example, each handwritten digit lies on a single manifold, but a smooth manifold covering all the digits does not exist. Since distance metric is non-metric, even cluster based approaches cannot guarantee that points similar to each other will fall in the same cluster.

We propose a way to split the data into a multi-level hierarchy, so that, local similarity property of dataset can be exploited to direct the search to correct branch of the tree while traversing it in top-down

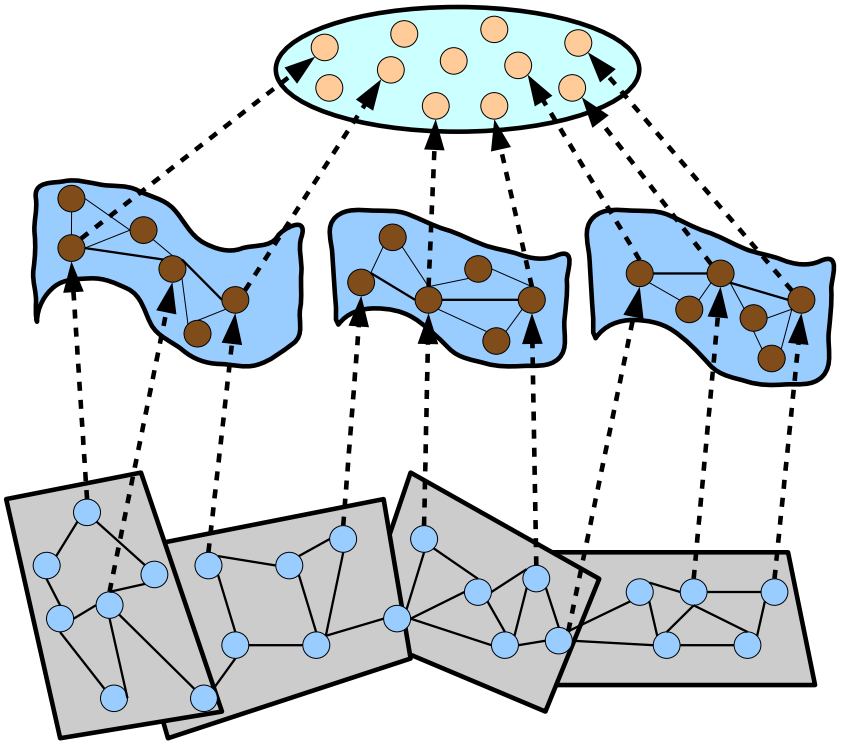


Figure 3.1: Hierarchical Local Maps.

fashion. We call this representation as *Hierarchical Local Maps(HLM)*. Using a tree structure for representation, helps in reducing the search space at each level and makes the algorithm scalable and incremental. New samples can be added in the hierarchy without modifying the existing tree structure. Such a tree representation, combined with a way to exploit local similarity, drastically reduces the explicit distance computations.

3.2 Construction of the Hierarchy

In a non-metric space, the computation of an optimal hierarchy for a given set of points, that minimizes the number of comparison needed for finding nearest neighbors is extremely difficult. Thus we use a greedy method to construct the hierarchy such that local neighborhood information gets embedded in a tree, which could be used later to direct the search to correct local maps. Let N denote the number of samples in training set S. The similarity function is denoted by F ; N_l and bf_l are the number of points and branching factor for each level, respectively. T denotes the minimum number of samples need to be present at top most level. Algorithm 1 describes the way to build the Hierarchical Local Maps.

In such a representation, a single point may lie on multiple nodes at a level in the hierarchy. Thus, if a point is overlooked at any level during a search, it could still be included in the levels to follow.

S : Training Set
N : Number of samples in training set S.
F : Similarity function
N_l : Number of points at level l
bf_l : Branching factor at level l
T : Minimum number of samples need to be present at top most level
Input : S, F, T
Output: $Tree, Num\_Levels$

$l = 1$
$N_1 = n(S)$
$New\_Set = \{\ \}$
while $N_l > T$ **do**
    $bf_{++l} = \lfloor P \log_{10}(N_l) \rfloor$
    while $\exists x \in S\ s.t.\ seen(x) == FALSE$ **do**
        Add an unseen point, $Seed$, to New_Set
        $nn \leftarrow$ bf-NN of $Seed$ in S according to F
        Mark $Seed$ and nn as seen
        Make $Seed$ parent of nn
    end
    $S = New\_Set$
    $New\_Set = \{\ \}$
    $N_l = n(S)$
end
$Num\_Levels = l$

Algorithm 1: Construction of HLM.

This is one of the major advantages this representation holds over the traditional tree-based search, where a misdirected search cannot be corrected. If the input space is metric, then this operation would preserve topology with zero distortion. However, in a non-metric space one might be able to find metric approximations of data points lying in a small neighborhood.

3.2.1 Parameter Selection for Construction of HLM

While constructing a hierarchy, T and bf_l needs to be tuned for any dataset. The T is a constant that is set empirically, based on the overall similarity of the points in the dataset (set at 50 in our experiments). As the total number of points in a level increases, the number of points similar to any given point also increases. Thus the branching factor at a level, is determined by the number of points in the level below. For our experiments, we set bf_l as:

$$bf_{l+1} = Plog_{10}N_l; \tag{3.1}$$

3.3 Nearest Neighbors Retrieval

To carry out a search, we need to describe a way to traverse down in the hierarchy. At the topmost level, we explicitly find the P_1 nearest neighbors of a query sample q using the similarity function F. The search process as described in Algorithm 2, requires only $P_1 \times P_2$ explicit distance computations at any level. The ISOMAP algorithm [45] is used to learn the manifolds formed by the nearest neighbors at each level. The embedding of a point is given by [17].

$$y = L_k^{\#} \left(\vec{\delta_a} - \vec{\delta_\mu} \right) \tag{3.2}$$

$$L_k^{\#} = \left[\frac{v_1^t}{\sqrt{\lambda_1}} \frac{v_2^t}{\sqrt{\lambda_2}} \frac{v_k^t}{\sqrt{\lambda_k}} \right]^t, \tag{3.3}$$

where $\vec{\delta_a}$ is squared distance between q and P_1 points, $\vec{\delta_a}$ is the mean of columns of the $P_1 \times P_1$ squared distance matrix. At every stage, the nearest $P_1 \times P_2$ points are determined in the low dimension, which are further reduced to P_1 using explicit computation of F. One can also include the similarity measures computed in the previous level for further refinement. If the search finds only $K1$ points at the last level of the hierarchy, where $K1 < K$, we expand the list by backtracking and finding more points at the previous level.

P_1 : Number of candidates chosen at each level
P_2 : Expansion factor in low dimension
q : Query point in original space
Num_Levels : Total number of levels in the hierarchy
Input : q, Num_Levels, P_1, P_2
Output: $NN : Nearest\ Neighbors\ List$

$level = Num\_Levels$
Put points of topmost level in S
$NN \leftarrow P_1$-NN of q in S
while $level > 1$ **do**
    Store children of NN in S_{child}
    Run Landmark Isomap to embed S_{child} with NN as landmarks
    Let Emb_{newpt} be embedding of q
    $K_1 = P_1 * P_2$
    Find K1-NN of Emb_{newpt} in low dimensional embedding
    Filter P_1 from $K1$ and update NN set
    Decrement level
end

Algorithm 2: Nearest Neighbor Retrieval.

3.3.1 Parameter Selection for Nearest Neighbor Retrieval

The NN retrieval algorithm has two parameters, P_1 and P_2. The application for which retrieval is being used determines their values. If for any application we need lesser percentage of nearest neighbors to be correct, then a lower value of P_1 can be chosen. However, if aim is at higher accuracy then a bigger value for P_1 and P_2 should be chosen. P_2 determines the weight given to distance computed in low dimensions.

3.3.2 Computational Complexity

In most applications, the non-metric distance computation is the most expensive operation to be performed. In the proposed search, the number of distance computations to be performed at the top-most level is around T. Further, at each lower level, we need to perform $P_1 \times P_2$ distance computations. Hence the overall computational complexity is $O(Num\_Levels \times P_1 \times P_2)$.

In addition to the above, we also need to compute $P_! \times P_2$ nearest neighbors from $P_1 \times bf_l$ samples in fixed dimensions at each level. The complexity of this process is $O(Num\_Levels \times P_1 \times bf)$, assuming the branching factor to be bf at all levels. The computation of the low dimensional space manifold requires the singular value decomposition of a P_1x$(P_1 \times bf)$ matrix, which is $O(P_1^2 \times bf)$ operations. The embedding process requires $O(P_1 \times d)$ multiplications, where d is the dimensionality of the manifold space. Note that our aim is to reduce the number of non-metric distances computed.

3.4 Experimental Results and Discussion

3.4.1 Unipen Handwriting Database

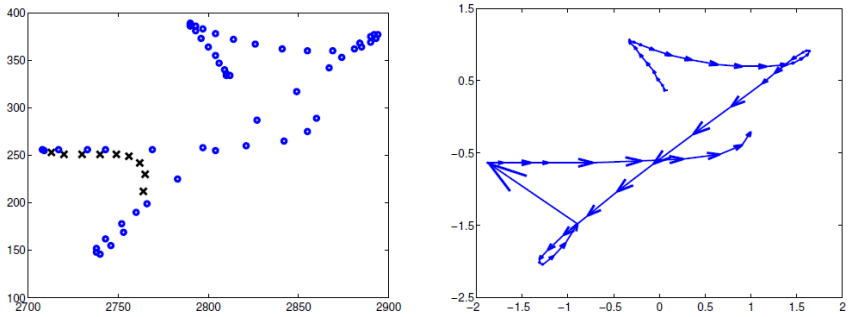


Figure 3.2: Left: Example of a seven. Circles denote pen-down locations, x's denote pen-up locations. Right: The same example, after preprocessing.

The online handwritten digit dataset that we use is the isolated digits benchmark of the UNIPEN Train-R01/V07 online handwriting database [24], which consists of 15953 digit examples. The digits have been randomly and disjointly divided into training and test sets with a 2:1 ratio (or 10,630 : 5,323

examples). We use the training set as our database, and the test set as our set of queries. Each query and database object in this dataset is preprocessed exactly as described in [5]. Figure 3.2 shows an example digit *seven* before and after preprocessing. The distance measure D used for classification is Dynamic Time Warping [32]. On an AMD Athlon 2.0GHz processor, we can compute on average 890 DTW distances per second. Therefore, nearest neighbor classification using brute-force search takes about 12 seconds per query. The nearest neighbor error obtained using brute-force search is 2.05.

To compare our results with BoostMap, we downloaded two distance matrices, one with DTW score between each pair of database object and other with DTW score between each test object and database object along with class labels for training and testing set from [2].

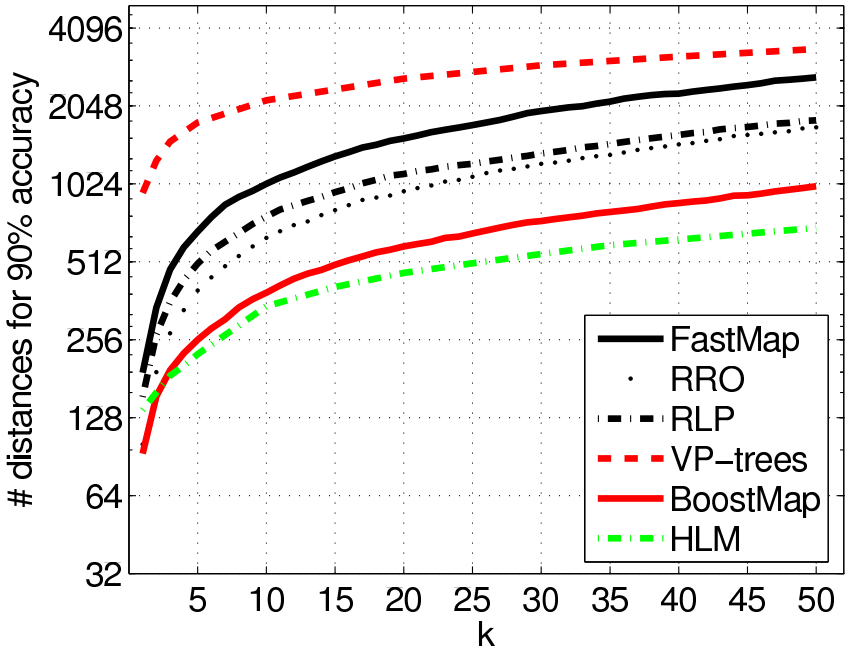


Figure 3.3: Number of DTW computations for K nearest neighbor retrieval for 90 percent accuracy.

During the construction of HLM, the value of P is set empirically as 2.5. We conduct several experiments for different values of accuracy one aims to achieve for different set of values for P_1 and P_2. The optimal value is obtained for $P_1 = 15$ and $P_2 = 1$. This means that the nearest neighbors computed in lower dimension are good enough and thus no refinement process is required.

The results of our algorithm is shown in Figure 3.3. As there is no external conditions or parameters of the dataset used, we directly used the values reported in the BoostMap paper [4] for other algorithms namely RRO, RLP, FastMap, VP-Trees[1]. Each subplot shows the exact number of DTW distances that needs to be computed against different values of nearest neighbors to be retrieved, for different accuracies on input dataset.

[1]We would like to thank Dr. Vassilis Athitsos, University of Texas, for providing the BoostMap Code and results for comparison.

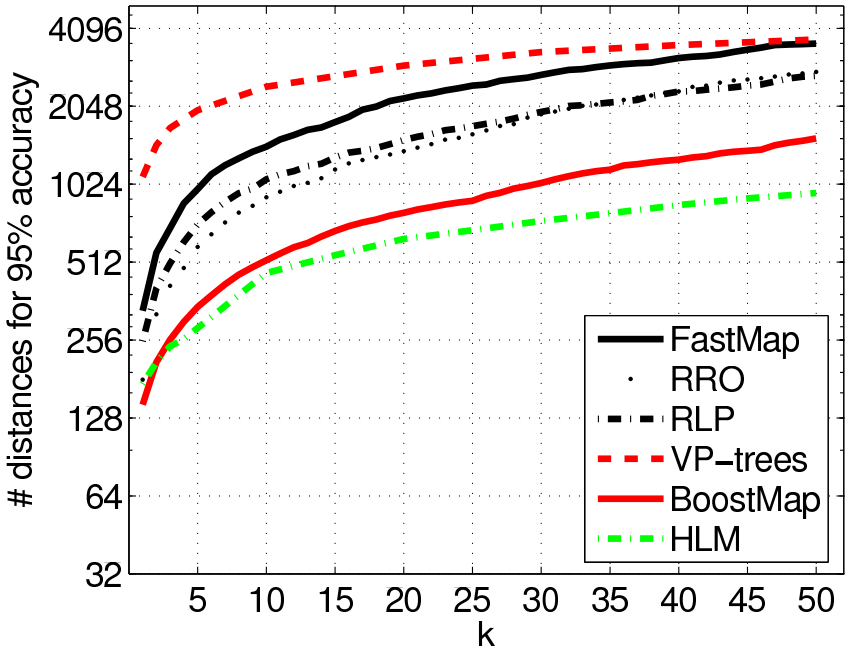


Figure 3.4: Number of DTW computations for K nearest neighbor retrieval for 95 percent accuracy.

For lower values of K, the difference in the number of distance computations is not significant but as K increases, HLM starts out-performing the other algorithms. Note that the values in Y-axis is in logarithm scale, so even small difference along Y-axis for higher K, signifies much more savings in terms of actual distances to be computed. To be precise, for 99% accuracy, BoostMap required 3302 exact distance computations, whereas HLM required only 1704 explicit distance measurements.

To study the nature of graph for $K > 50$, we find the number of nearest neighbors that can be retrieved with respective accuracy for the same number of distance computations required for $K = 50$ in BoostMap. For 90% and 95% accuracy, HLM can find 127-NN which is 2.5 times the nearest neighbors found by BoostMap using same number of distance computations. For 99% accuracy, HLM can find 110-NN. Thus we can say that the nature of the graph would be the same for higher values of K.

Another measure of the approximate nearest neighbors computed is their similarity to the query point. To evaluate this, we perform k-NN classification using the approximate NNs and compared with accuracy achieved using exact NN from direct DTW measurements. For the chosen parameter, we are left with around 30 points in last level after calculating around 130 DTW distances. Hence, the total number of DTW distances to be computed in order to find the first nearest neighbor (approximate) is around 160.

Table 3.1, shows the classification accuracies for different values of k, when we use HLM instead of computing DTW distances to all points. As indicated by the first column, our approach can classify a sample within 0.5% accuracy of the ideal case, while achieving a 98.5% savings on DTW distance computations (160 instead of $10,630$).

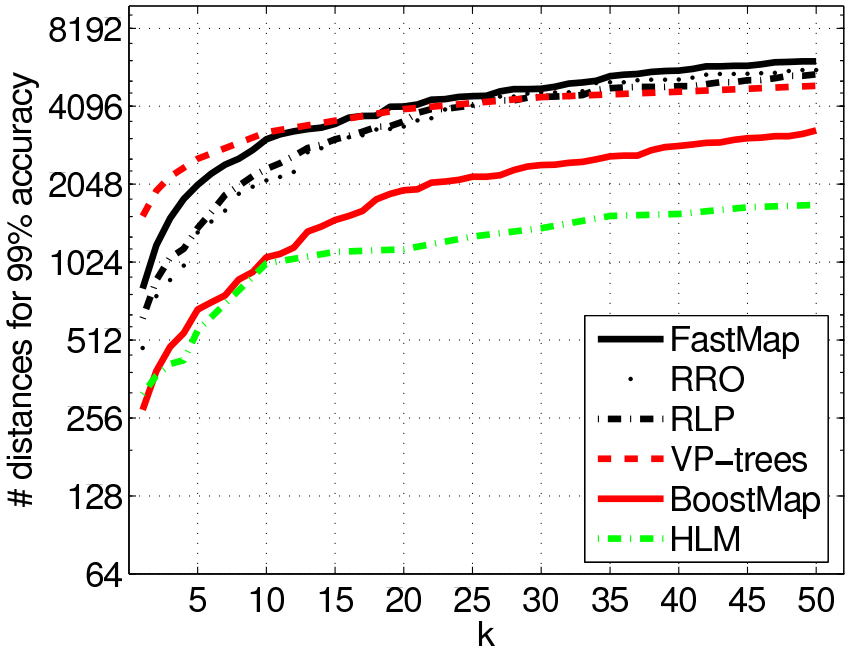


Figure 3.5: Number of DTW computations for K nearest neighbor retrieval for 99 percent accuracy.

k	1	5	10	15	20
DTW	98.1	97.73	97.41	96.99	96.83
HLM	97.65	97.27	97.08	96.69	96.44

Table 3.1: Classification Accuracy on UNIPEN Dataset using exact and approximate k-NN.

3.4.2 CASIA Iris Database

However, the nature of matching score using for biometric identification or verification is highly discriminative, with small intra-class distances and larger inter-class distances. Hence the matching score cannot be used for computing approximate nearest neighbor. For strong biometric traits such as the iris pattern, the similarity score between any two samples belonging to different classes will be close to each other, making the hierarchical search almost random. To solve this problem, one should use a smoother distance function in the HLM construction and retrieval. Note that the problem in the case of iris based identification is not that of high computational cost of the distance metric, but the sheer size of the database, which makes explicit comparison with every sample, impractical.

For iris based person identification, we segment the iris pattern into a set of concentric circles as seen in bottom two rows of Figure 3.6, and each circle further into sectors. We characterize each segment using the average gray value, after normalization of the whole image, resulting in a 160 dimensional feature vector. Euclidean distance between two such vectors is used to find the approximate nearest neighbor. We use the term *soft metric* to refer to this distance measure, as opposed to the matching score

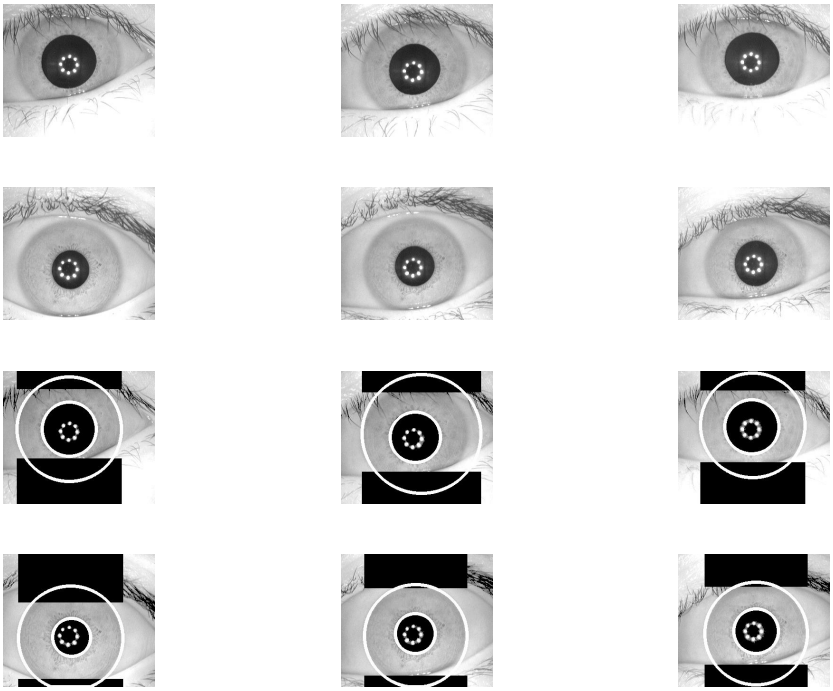


Figure 3.6: Top two rows correspond to iris of two persons. The bottom two rows shows the concentric region segmented from the corresponding iris image.

used for the biometric. To compare the results, we also perform the experiments with the matching score computed using the 20x240 feature vector, proposed in [16]. Matching score is measured by hamming distance between two bit streams as follows:

$$HD = \frac{\parallel (codeA \otimes codeB) \cap maskA \cap maskB \parallel}{\parallel maskA \cap maskB \parallel} \tag{3.4}$$

Experiments are conducted on the CASIA Iris Image Database V3.0 [1]. For experimental evaluation, the CASIA-IrisV3-Interval was used as it contains the larger number of images, captured in two different sessions. Database consists of left and right eye images of 249 subjects. Six images per eye of a subject are randomly chosen and divided equally in training and testing set. We discard those users for which less than six images per eye were present. In total 855 images were present in training and testing set, corresponding to 285 eyes, with three samples per eye. Top two rows of Figure 3.6 shows sample iris images of two users from the CASIA dataset. We construct the HLM structure using the distance measure mentioned above.

For the construction of HLM, as samples of most of the class will not be present in the top most level, we set T to be a constant: 50 (irrespective of number of classes). Figure 3.7 shows the variation

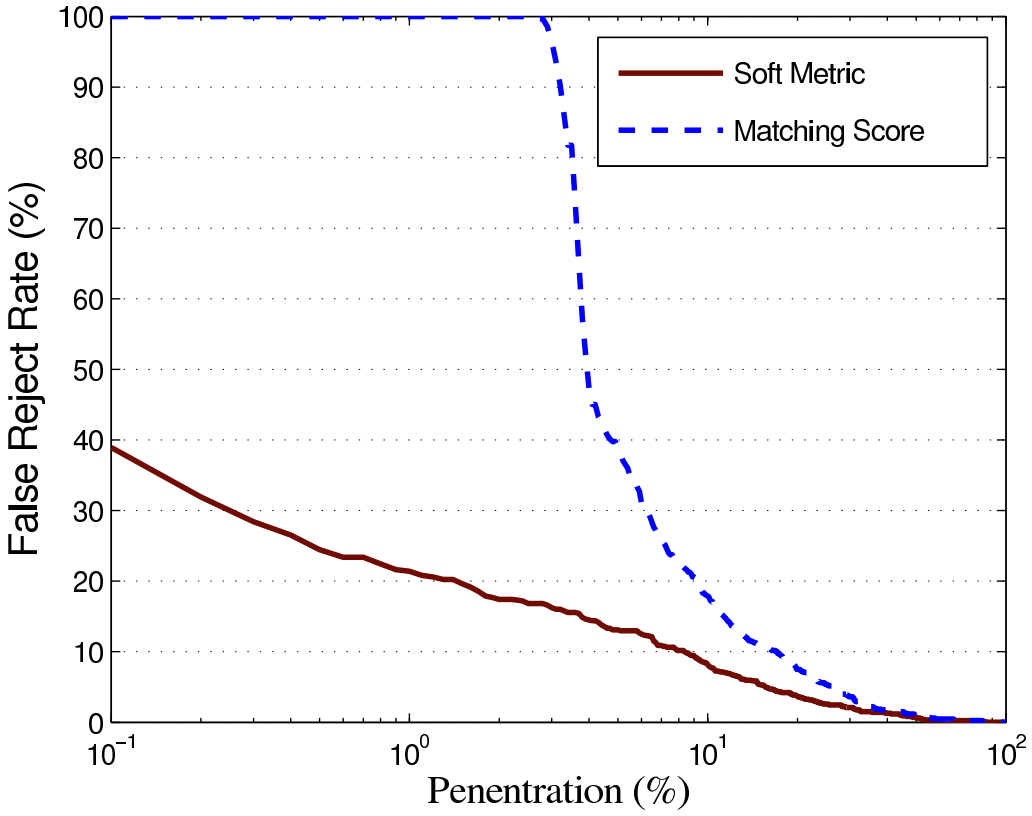


Figure 3.7: FRR vs Penetration (CASIA Iris).

of False Reject Rate(FRR) with Penetration Rate for $P_1 = 15$ and $P_2 = 1$. We note that one can find a matching pattern at the first nearest neighbor in around 60% of the case, which requires around 40 soft metric computations and 75 distances in a 15-dimensional subspace. Note that this is very small compared to the 855 matching scores to be computed for the brute force approach. Moreover, finding of the following nearest neighbors take only around 1 comparison on an average.

On the downside, a large number of similarity measures need to be evaluated during offline stage of constructing the HLM. The storage complexity is also linear in the number of training samples. We note that the search works better when used with smoother similarity measures, which are also well correlated with the original distance metric, F.

Chapter 4

Hierarchical Local Map and Biometric Data

Biometric authentication provides a convenient and reliable to estibilish one's identity, which can be extremely useful in a variety of day-to-day activities such as banking, physical access control to computing resources, etc. Recent advancements in biometric sensors and matching algorithms have led to deployment of biometric authentication in a large number of civilian and government applications. For a biometric based system to be online, the response time and search and retrieval efficiencies become important in addition to accuracy. Along with these attributes, machines with high computation speed should not be a bottleneck in deployment of such systems(server). The server should be able to perform identification and verification in less time or else waiting time for queries present in queue would increase.

In practice, most biometric identification systems function by explicitly comparing a query biometric with each biometric template stored in a database. The main goal in biometric indexing is to reduce the number of templates to be considered for a match by identification system. The set formed after the filtering phase is called the *candidate set* or the reduced search space. Hit rate is defined as the probability of a possible match in the candidate set. The efficiency of the indexing scheme can be measured in terms of both hit rate or penetration rate, which is defined as the fraction of data in the candidate set that are compared to classify the query template correctly. In the previous chapter, we proposed a gradient indexing method, where indexing is formalised as an approximate ordering problem. In order to find a match for the test template (probe), the samples would be arrange in decreasing order of their similarity with the probe. This order can now be used to tackle two kind of problems. Consider the scenario, where time is a constraint during the identification process. Our approach would guide the search to the more probable samples first, and thus probability of finding a match is high for a given time constraint. Consider another scenario where a threshold value of similarity is empirically known to system. For this case, search would continue till a sample is found having similarity higher than the precomputed threshold. Pushing samples similar to probe higher in search order will facilitate the search to stop much earlier than it would been if the search is random.

In this chapter, we look at the problems of indexing in high dimensions, and see the relation between a high dimensional data and biometric data. We define the term *softness/hardness* for a biometric distance

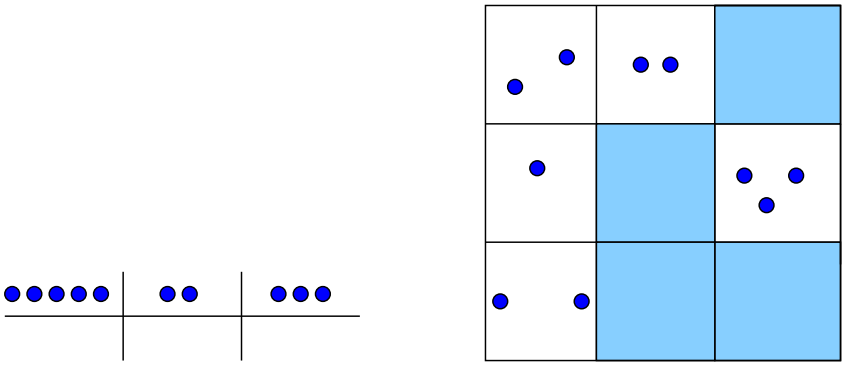


Figure 4.1: Figure shows the distribution of points in bins if each dimension is split in three regions. It can be seen number of free cells decreases rapidly with growing dimensions.

measures. We study the performance of the proposed Hierarchical Local Maps(HLM), on different datasets, and see how various parameters affect its indexing performance.

4.1 Problems in high dimension

As seen in the Section 2.2, the effectiveness of all traditional index structures are reduced owing to the curse of dimensionality, (see Figure 2.5). It is a significant obstacle in high dimension data analysis, which refers to the fact that a local neighborhood in higher dimensions is no longer local, or to put it another way, the sparsity increases exponentially given a fixed amount of data points. In general terms, problems with high dimensionality result from the fact that a fixed number of data points become increasingly sparse as the dimensionality increases. In high dimensions one can be see that the volume of a hypersphere inscribed inside a hypercube converges to zero. Thus the search for the nearest sample obtain no answer even when radius of search is taken to half of the length of hypercube. To visualize this, consider 100 points distributed with a uniform random distribution in the interval [0, 1]. If this interval is broken into 10 buckets, then it is highly likely that all buckets will contain some points. Next we distribute the same number of two dimensional over a unit square. If we keep the unit of discretization to be 0.1 for each dimension, then we have 100 two-dimensional regions, and it is quite likely that some regions will be empty. For 100 points and three dimensions, number of regions would increase to 1000. Since number of regions are far greater than the number of points, most of the region will be empty. Conceptually the data is lost in space as we go to higher dimensions. Figure 4.1 shows the distribution of points in bins if each dimension is split in three regions. It can be seen number of free cells decreases rapidly with growing dimensions.

All space partitioning schemes, partition the data space to perform indexing. Each partition has a fixed region described by its lower bound and upper bound, which signifies the boundary for an object lying in that partition. A typical search is performed in two phases, namely filtering and refinement

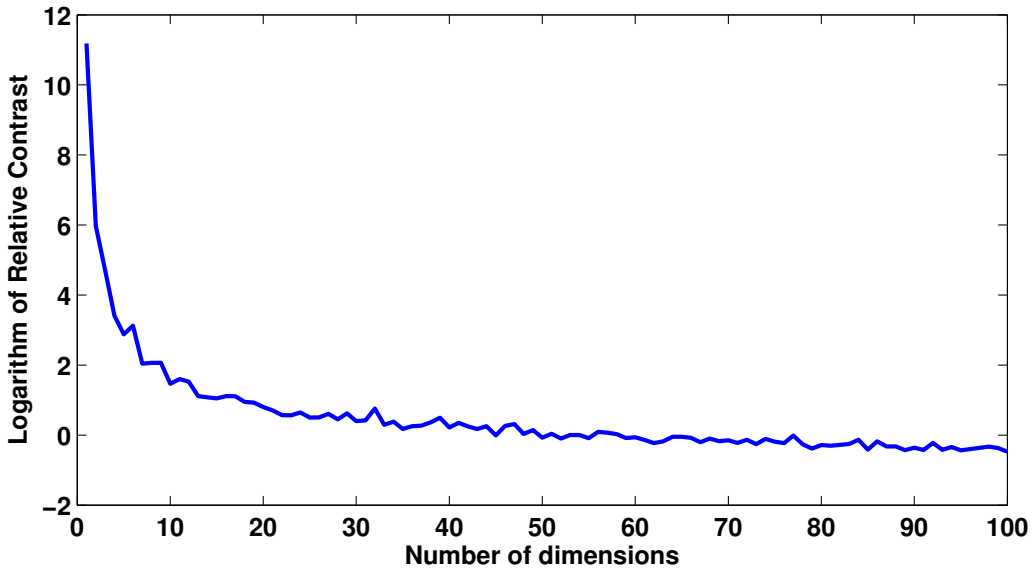


Figure 4.2: Data Distribution in High Dimensions

phases. In the former phase, the index structure is scanned, to filter irrelevant partitions and thus obtain a candidate set. In the latter phase, candidate subsets of the original data file are accessed. The performance of indexing scheme is highly dependent on the filtering phase, in which irrelevant partitions are excluded from the candidate list.

This mechanism wold work well if the number of objects to index are far larger than that of number of partitions, i.e., the density of points in each partition is uniform. But this assumption collapses when the dimensionality of the object becomes very high. As the dimensionality of space increases, the number of partitions grows exponentially, and thus density of points falling in each partition decreases exponentially. In such a scenario, the filtering phase is no longer meaningful. Also as the dimensionality increases, the data space cannot be divided clearly into tree nodes. Thus, subspaces corresponding to most nodes overlap each other and the pruning power is lost.

The concept of similarity/dissimilarity between points no longer holds when the space dimensionality increases as all the points are equi-distant to each other. In other words, the relative contrast of the distances approaches zero, as the dimensionality of the data increases.

$$lim_{D\to\infty}\frac{Distmax - Distmin}{Distmin} \to 0. \tag{4.1}$$

Figure 4.2 shows the plot of logarithm of relative contrast against number of dimensions for 100 points. Euclidean distance is taken as the measure of distance between two points.

4.2 Relation between biometric data and high dimensional data

To study the relation between biometric data and high dimensional data, we first introduce the concept of degree of softness of a biometric measure.

Softness/Hardness of a biometric measure

Softness is defined as the measure of separability in genuine and imposter scores. If for a biometric datasets, genuine score and imposter scores are well separated, we call it a hard biometric. Iris and finger print are considered to be hard biometric as there is a clear separation in the genuine and imposter scores. For two samples of same class, the matching score returned by hard biometric would be close to 1, where as for two samples of different class, the matching score would be close to 0.5. On the other hand, if there is no proper defined boundary between genuine and imposter scores, we call it a soft biometric. Naturally, soft biometric are bad for classification purposes when compared to hard biometric.

Thus, points in high dimensions can be viewed as samples from different class where matching score can be mapped to the distance between points in this space. Each point is at the same distance from each other and thus variance of the inter-point distance is very low. This corresponds to low variation in the imposter scores. Now that, we see a correspondence between high dimensional data and a biometric data, the degree of softness for a biometric data can be defined as the ratio of between class and within class scatter.

Daugman [16] conducted an experiment on the 4258 different iris images to study the distribution of imposter scores. The histogram in Figure 4.3 shows the distribution of matching score (Hamming Distance) obtained from 9.1 million comparisons. It closely fits a binomial distribution with 249 degrees of freedom. The observed mean of imposter scores was 0.499 with a standard deviation of 0.0317. Intuitively, if we see, as there is equal probability for a bit of iris feature vector to be 0 or 1, the expect proportion of matching bits would be equal to 0.5, which is close to the mean of the imposter scores. For the genuine scores, the mean was found out to be 0.110 with a standard deviation of 0.065.

A low value of standard deviation denotes that there is little variation in the genuine and imposter scores, and with the means well separated, it can be considered as a hard biometric.

As seen in Figure 3.7, though a hard biometric is good for classification purposes, the indexing performance of such a measure is very poor, as the nature of similarity score distribution highly correlates with the distance distribution in high dimensions, where owing to curse of dimensionality indexing performance drastically reduces. Use of a softer distance measure, like DC component of the gabor responses works better for indexing than the hard biometric measure by computing hamming distance on the daugman's iris feature vector. Thus a softer measure in the first step (filter phase), would discard most of the irrelevant classes and a candidate set is obtained which can be expanded incrementally. In the refine phase, using the hard biometric measure, the candidates are scanned sequentially to obtain the correct match.

As stated earlier, points in very high dimensions would correspond to hard biometric as variance of inter-point distance would be too low. As the number of dimensions decreases, variance of inter-point

distance starts increasing, and maps to a softer biometric. To generate the biometric data with different softness, we synthetically generate the data by varying its dimension and ratio of within class to between class ratio. We study the indexibility of our method on this synthetic datasets.

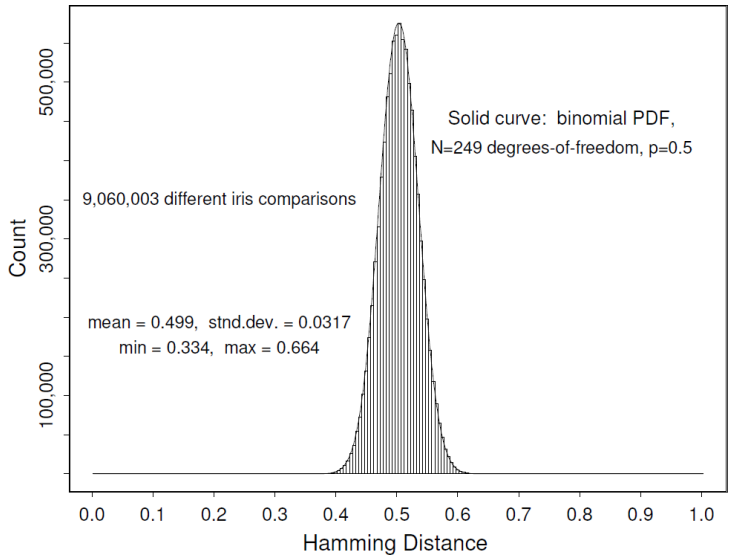


Figure 4.3: Distribution of Hamming Distances from all 9.1 million possible comparisons between different pairs of irises in the database. The histogram forms a perfect binomial distribution with p = 0.5 and N = 249 degrees-of-freedom [16]

4.3 Experiments on synthetic data set

Experiments were conducted on synthetic datasets to study the performance of HLM on the degree of softness of a distance measure. First of all, mean of each class point was sampled from a 1-D gaussian with mean 0 and sigma 1. To generate a d-dimensional point, the sampling was performed d times, once for each dimension. Now with each of the class mean as center, same class points were generated by sampling from a gaussian with mean as the class mean and varying sigma. Again treating each dimension as independent of other, sampling was performed d times to obtain a d dimensional point. Number of points in a class was kept to a constant value of 10 for training data and 5 for testing data. We study the performance, on changing two parameters:

- Within class standard deviation
- Dimensionality of the point in original space

4.3.1 Indexing performance varying number of dimensions

Experiments were conducted to study the affect of number of dimensions on the indexing performance in high dimensions. Penetration rate is chosen as the criteria for measuring indexing performance. Number of classes in training and test were set to 500. We measure the penetration rate for 90 percent of test samples to be classified correctly. We synthetically generated data of dimensionality varying from 100 to 400 in step of 100. Within class standard deviation was set to 0.2.

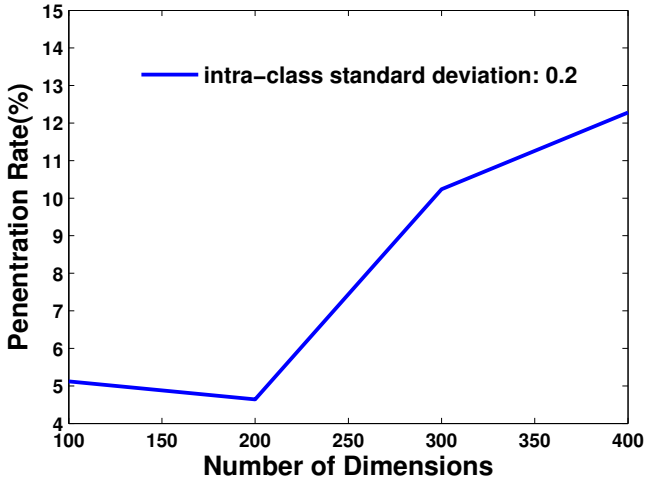


Figure 4.4: Penetration Rate vs Number of dimensions for 90 percent accuracy on test samples.

Figure 4.4 shows the plot between penetration rate and number of dimensions for 90 percent accuracy on test samples.

As it can be seen, as number of dimensions increases penetration rate goes down up to a certain point but then again rises after a point, owing to curse of dimensionality. Our goal would be to reduce the penetration rate for the area where the indexing scheme starts breaking. Next we look, if increasing the within class standard deviation, i.e. making the measure a little softer, can reduce the penetration rate.

4.3.2 Indexing performance varying within class to between class variance ratio

We conducted experiments with varying intraclass standard deviation, keeping interclass variance to constant value of 1. To study the affect of the ratio of within class to between class variance ratio in high dimensions, we synthetically generated data of dimensionality varying from 100 to 400 in step of 100. Indexing performance is measured for 90 percent accuracy of test samples to be classified. Figure 4.5 shows the plot between dimensionality of data and penetration rate of data i.e., percentage of data to be looked for 90 percent accuracy on test samples. We plotted the trend for 4 different values of intra-class standard deviation: 0.2, 0.4, 0.6 and 0.8. We show the trend for the case when number of classes were kept to 500. As described earlier, the plot corresponding to lower value of standard deviation would correspond to harder biometric measure. As it can be seen in the figure, as the dimensionality increases,

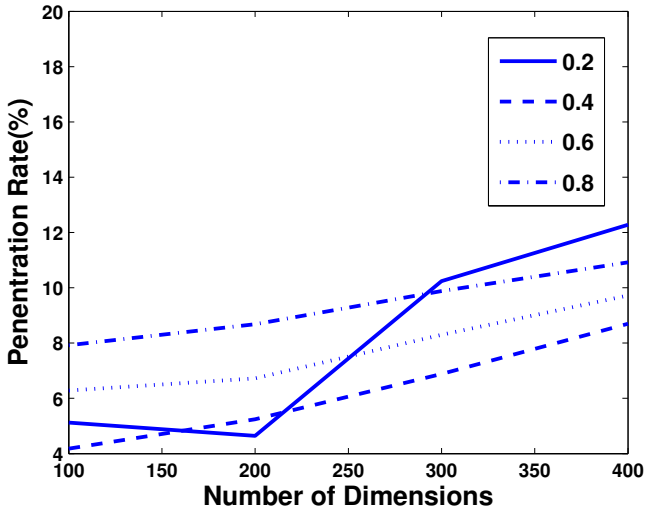


Figure 4.5: Penetration Rate vs Number of dimensions for 90 percent accuracy on test samples.

the graph corresponding to 0.2 starts increasing gradually and in the process rises above the graph corresponding to 0.8. Thus, a hard biometric would have a high penetration rate in high dimensions as compared to softer measures. Also, the number of lookups required for intraclass standard deviation 0.4 is lower than others for higher dimensions even though its softness is less as compared to 0.6 and 0.8. Thus, we see indexing performance is highly dependent on the dimensionality of the data. Ratio of within class to between class standard deviation, can give an insight into the performance of the indexing scheme for a fixed dimension.

4.4 Conclusion

Indexing biometric data is a challenging problem as there is no natural order in such databases. Points in high dimension behave in a same way as the biometric data as for both the relative contrast tends to zero. We see, use of a softer measure, like using DC coefficient in case of iris dataset, would increase the indexing performance even though its not good for classification. It is due to the fact, by making a hard biometric softer is same as reducing the dimensionality of data so that the indexing scheme becomes effective. Also, we conclude the degree of softness required for a data for better indexibility is highly dependent on the dimensionality of the data. Use of a softer biometric in filtering step to form a candidate set, which could be expanded incrementally using Hierarchical Local Maps, and then using the corresponding hard biometric in the refinement step would decrease the computation time for classification; when compared to applying a hard biometric directly on the dataset.

Chapter 5

Conclusions and Future Work

We presented a novel approach for robust approximate nearest neighbor retrieval for computationally expensive distance measures. We reduce the number of exact distance computations done by a computationally intensive distance measure for k-nearest neighbor retrieval and classification. Indexing and classification can often be seen as equivalent problems. Finding the nearest neighbors of the query can be seen as a classification problem, where we need to determine, for each database object, whether it is a nearest neighbor of the query or not. At the same time, classifying an object, regardless of the classification method we use, can be seen as a nearest neighbor problem where the database is a set of classes and we want to find the class that is nearest to the object. A way to arrange the data in multiple levels of hierarchy is proposed so that local neighborhood information can be utilized during search to guide it to correct local map. The use of hierarchical structure helps in discarding the portion of data that is irrelevant to query and speeding up the process. Currently, parameters used for tree construction are set empirically in offline step. Retrieval as well as classification is done using filter and refine strategy. Refinement step is applied at each level of the hierarchy too while traversing it from top to bottom, leading to better hit rate in candidate set. Results of k-nearest neighbor retrieval as well as classification results on UNIPEN dataset shows the advantages of using HLM over state-of-the-art approximate nearest neighbor retrieval algorithms.

The method is scalable as well as incremental. we provide a promising step forward towards developing a general indexing framework for non-metric spaces whose geometric structure is either poorly understood or inconvenient from the point of view of existing indexing methods. We obtain a method that is free from geometric assumptions, is equally principled in metric and non-metric spaces, and can capture non-metric structure.

Non-Euclidean and non-metric computationally expensive distance measures are frequently utilized in computer vision, and we have demonstrated that the proposed methods can be successfully applied in a variety of domains to achieve state-of-the-art accuracy and efficiency at the same time.

We also looked on the property of data that would favor indexing. We inferred, indexing of the biometric data can be done in two steps. The first step involves, building and training HLM using a softer biometric measure, i.e., filtering step uses softer biometric measure to build a candidate set,

which is expansible. The second step involves refinement using the hard biometric measure; as when there will be a hit in the identification process, the search would stop. Thus a soft biometric inspite of bad for classification purpose, can help in decreasing the penetration rate by reducing the candidate step to be refined by the hard biometric. Classification result on CASIA iris dataset by using average gabor response for a block as the feature vector along with Euclidean distance as the soft biometric measure in conjugation with Daugman's feature vector and hamming distance as the hard biometric shows the advantage of using a softer metric over a hard metric for indexing.

One interesting extension of the work would be to learn a function that would return the degree of indexibility for a data without actually running an indexing scheme on it. As we know, indexing scheme are dependent on the data distribution and the similarity function; we could extract the parameters from data distribution and similarity function that effects the indexing performance and learn the function. If derivative of such a function exists, we may apply techniques like gradient descent or Expectation Maximization, to obtain a global minima on the function, that would work best for a particular data. One of the potential directions of improving the algorithm would look into optimal construction of HLM. One could also extend the applicability of the approach by defining similarity measures that allow hierarchical representation.

Related Publications

Pratyush Bhatt and Anoop Namboodiri **Hierarchical Local Maps for Robust Approximate Nearest Neighbour Computation** *7th International Conference on Advances in Pattern Recognition (ICAPR 2009)*, Feb. 4-6, 2009, Kolkotta, India.

Bibliography

[1] $http://www.cbsr.ia.ac.cn/IrisDatabase.htm$.

[2] $csr.bu.edu/asl/data/bm_datasets/unipen/$.

[3] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff. Simultaneous localization and recognition of dynamic hand gestures. In *In Proc. IEEE Workshop on Motion and Video Computing*, pages 254–260, 2005.

[4] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Boostmap: An embedding method for efficient nearest neighbor retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(1):89–104, 2008.

[5] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines : A kernel approach. In *IWFHR '02: Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, page 49, Washington, DC, USA, 2002. IEEE Computer Society.

[6] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: two new techniques for image matching. In *IJCAI'77: Proceedings of the 5th international joint conference on Artificial intelligence*, pages 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[7] R. Bellmann. Adaptive control processes. In *Princeton University Press, Princeton, NJ*, 1961.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:509–522, 2001.

[9] S. Berchtold, C. Böhm, and H.-P. Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 142–153. ACM Press, 1998.

[10] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree : An index structure for high-dimensional data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 28–39. Morgan Kaufmann, 1996.

[11] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

[12] B. Boeckmann, A. Bairoch, R. Apweiler, M. claude Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, I. Phan, R. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res*, 31:365–370, 2003.

[13] Borst, Thurler, Breant, Lehner-Godinho, Calmy, and Meier. Finding similar cases within a hospital information system. *Studies in health technology and informatics*, 77:875–879, 2000.

[14] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:529–533, 1995.

[15] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley, 2 edition, 1991.

[16] J. Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14:21–30, 2004.

[17] V. de Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. *Stanford Mathematics Technical Report*, 2004.

[18] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[19] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In M. J. Carey and D. A. Schneider, editors, *SIGMOD Conference*, pages 163–174. ACM Press, 1995.

[20] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[21] A. W.-C. Fu, P. M. shuen Chan, Y.-L. Cheung, and Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *VLDB J.*, 9(2):154–173, 2000.

[22] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[23] I. Guyon, L. Schomaker, and R. Plamondon. Unipen project of on-line data exchange and recognizer benchmarks. *Proc. of ICPR*, pages 29–33, 1994.

[24] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *In Proceedings of International Conference on Pattern Recognition*, pages 29–33, 1994.

[25] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[26] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network, 2004.

[27] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):530–549, 2003.

[28] D. P. Huttenlocher, G. A. Klanderman, G. A. Kl, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.

[29] A. Hyvrinen. Independent component analysis. *Neural Computing Surveys*, 2, 2001.

[30] I. Jolliffe. *Principal Component Analysis*. Springer, New York, NY, 1986.

[31] E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.

[32] J. B. Kruskal and M. Liberman. The symmetric time-warping problem: from continuous to discrete. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules - The Theory and Practice of Sequence Comparison*. CSLI Publications, Stanford, CA 94305, 1999.

[33] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[34] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high-dimensional data. *VLDB J.*, 3(4):517–542, 1994.

[35] R. Micheals, P. Grother, P. Grother, R. J. Micheals, P. J. Phillips, and P. J. Phillips. Face recognition vendor test 2002 performance metrics. In *Proceedings 4th International Conference on Audio Visual Based Person Authentication*, pages 937–945. Springer, 2003.

[36] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[37] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *NIPS*, pages 536–542. The MIT Press, 1998.

[38] A. Moore. A tutorial on kd-trees. 1991. Available from http://www.cs.cmu.edu/simawm/papers.html.

[39] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007.

[40] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.

[41] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[42] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[43] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *In ICCV*, pages 750–757, 2003.

[44] A. Smellie. Accelerated k-means clustering in metric spaces. *Journal of Chemical Information and Modeling*, 44(6):1929–1935, 2004.

[45] J. B. Tenenbaum, V. Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[46] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[47] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2000.

[48] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. J. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *KDD*, pages 216–225. ACM, 2003.

[49] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205. Morgan Kaufmann, 1998.

[50] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 62–73, 1996.

[51] C. K. I. Williams. On a connection between kernel pca and metric multidimensional scaling. *Machine Learning*, 46(1-3):11–19, 2002.

[52] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. *CoRR*, abs/1004.5370, 2010.