Density-Based Indexing for Approximate Nearest-Neighbor Queries

Kristin P. Bennett*
Rensselaer Polytechnic Inst.
bennek@rpi.edu

Usama Fayyad
Microsoft Research
fayyad@microsoft.com

Dan Geiger†
Technion, CS Department
dang@cs.technion.ac.il

October 28, 1998

Abstract

We consider the problem of performing nearest-neighbor queries efficiently over large high-dimensional databases. Assuming that a full database scan to determine the nearest neighbor entries is not acceptable, we study the possibility of constructing an index structure over the database. It is well-accepted that traditional database indexing algorithms fail for high-dimensional data (say $d > 10$ or 20 depending on the scheme). Some arguments have advocated that nearest-neighbor queries do not even make sense for high-dimensional data since the ratio of maximum and minimum distance goes to 1 as dimensionality increases. We show that these arguments are based on over-restrictive assumptions, and that in the general case it is meaningful and possible to perform such queries.

We present an approach for deriving a multidimensional index to support approximate nearest-neighbor queries over large databases. Our approach, called DBIN, scales to high-dimensional databases by exploiting statistical properties of the data. The approach is based on statistically modeling the density of the content of the data table. DBIN uses the density model to derive a single index over the data table and requires physically re-writing data in a new table sorted by the newly created index (i.e. create what is known as a clustered-index in the database literature). The indexing scheme produces a mapping between a query point (a data record) and an ordering on the clustered index values. Data is then scanned according to the index until the probability that the nearest-neighbor has been found exceeds some threshold. We present theoretical and empirical justification for DBIN. The scheme supports a family of distance functions which includes the traditional Euclidean distance measure.

Microsoft Research
Technical Report MSR-TR-98-58

Revised: February 28, 1999

Contact Author:	Usama Fayyad (http://research.microsoft.com/~fayyad)
address:	Microsoft Research
	One Microsoft Way
	Redmond, WA 98008, USA
phone:	+1-425-703-1528
fax:	+1-425-936-7329
e-mail:	fayyad@microsoft.com

*This work was performed while the author was visiting Microsoft Research
†This work was performed while the author was on sabbatical at Microsoft Research

1 Introduction

An important class of queries in data mining applications is a class of queries known by several names including: *similarity queries, nearest-neighbor queries*, and *similarity joins* [28]. The basic problem is: given a data table of records, find the nearest (most similar) record(s) to a given query record. More specifically, the problem we consider is: Given a data record represented as a d-dimensional vector of values q, and a data set D, determine the record $x \in D$ such that

$$\text{Dist}(x, q) = (x - q)^T S(x - q)$$

is minimized, where S is a positive semi-definite matrix. For example, if the distance measure happens to be the Euclidean distance, also known as the 2-norm, the S is the identity matrix, and the distance has the form:

$$\text{Dist}(x, q) = \sqrt{\sum_{i=1}^{d} (x_i - q_i)^2}$$

This can be generalized to retrieving the K nearest records. Other distance norms can be used, possibly with weights W_i associated with each dimension. The obvious approach to finding the nearest neighbors is to scan the data and compute the distance between every record in the database and the query record. With large databases, such a solution is not acceptable and it would be desirable to provide an index structure that restricts the set of records to be scanned. This topic has been the subject of study by several research communities.

Applications of nearest-neighbor queries are many and include: predictive modeling (e.g. in marketing or medical diagnosis), product catalog navigation, advertising (especially on-line), fraud detection, customer support, problem diagnosis (e.g. for product support at service centers when people call with queries), and management of knowledge bases. It is also a method for providing more flexible means for querying databases by supporting a "find similar" capability. Such capabilities are useful in text or image databases. With the growth of Data Warehousing, nearest-neighbor queries are also useful for flexible querying as well as data cleaning applications. Of course with large databases, the problem of effective indexing gains importance. There is a fairly large literature on this topic in the database and the pattern recognition literatures. Work in pattern recognition has primarily focused on the type of distance measure used (sometimes called nearest-neighbor norms) or on in-memory data structures for optimizing retrieval [11].

Work in the database literature has focused primarily on efficient data structures for retrieving matches from disk. It is a well-accepted fact in the database literature that the traditional indexing methods fail to be useful if the dimensionality of the data is high [25, 20, 22, 5, 3, 1, 31, 23, 27]. Since a full data scan to check all distances is ruled out for large databases, another solution is necessary when the data is high dimensional. There is increasing interest in avoiding this "curse of dimensionality" by performing *approximate* nearest-neighbor queries [21]. In data mining applications, the nearest-neighbor distance metric is typically a mathematical approimation of users, usually aesthetic, similarity criterion. Thus there is little

benefit in performing exact nearest-neighbor queries when faster approximate nearest neighbor methods are available.

Notably, a recent work [6] questions whether a nearest-neighbor query in high dimensions makes sense at all under commonly used assumptions on the distribution of data. We discuss this argument later in the paper, but in essence the argument in [6] makes the claim that as dimensionality increases, the ratio of distance between the closest and farthest elements to a query point goes to 1. We show that this argument relies on assumptions that are not necessarily true for most databases. We show that under more realistic assumptions we can guarantee that the points are well separated. This allows us to develop an approach for building a multidimensional indexing scheme and characterize when it will be most efficient.

1.1 Summary of Approach

The approach proposed in this paper is based on the idea of using a statistical model of the contents of a data table as a basis for defining a new multi-dimensional indexing scheme which supports **probabilistic** or **p-approximate nearest-neighbor (p-NN) queries**. A point s is the p-NN of q if with probability greater than $1-p$, s is the true NN of q. Note p-NN differ from the prior $\epsilon-$approximate nearest-neighbor (ϵ-NN) [21]. The ϵ-NN queries find points within ϵ of the true nearest neighbor distance. A point s is the ϵ-NN of q if for all $x \in D$, $dist(s,q) \leq (1+\epsilon)dist(x,q)$. To calculate p-NN queries requires a statistical model. The statistical model takes the form of an estimate of the probability density function (pdf) of the data in the multi-dimensional space defined by the attributes. We discuss the form of this pdf and related issues in Section 3.

Assuming we have a statistical model of the data, how does it help us with constructing an indexing scheme for nearest-neighbor queries? As we discuss in Section 2, most indexing methods based on carving the data space into box regions fail in high dimensions due to high overlap between a query region (for points in the neighborhood of a given query point) and the box regions defined by the index. In high-dimensional spaces, a query region can easily overlap with most index regions. Most indexing methods must examine any region that intersects the query region. What would help in such instances is the ability to *prioritize* regions and *rule out* a subset of them as highly unlikely to contain data records relevant to the query. This is exactly the intuition behind the method we propose. Of course, with a statistical model, the determination of a region's likelihood to contain the nearest neighbors to a query point will necessarily be probabilistic. However, depending on the data and the density estimate obtained, it may be possible to rule out or rule in regions with high enough probability. We address the issue of how to obtain the density estimate in Section 3.1 and we cover the details of the algorithm in Section 4. We provide a basic outline of the method here.

The method we use for estimating the density and then constructing the new clustered index is obtained using *data clustering techniques*. The data clusters obtained from a scalable clustering implementation are used to decide the ordering of the records in the table. Essentially the records need to be sorted by their cluster ID and the table materialized. This table will be scanned via a clustered index for the purposes of

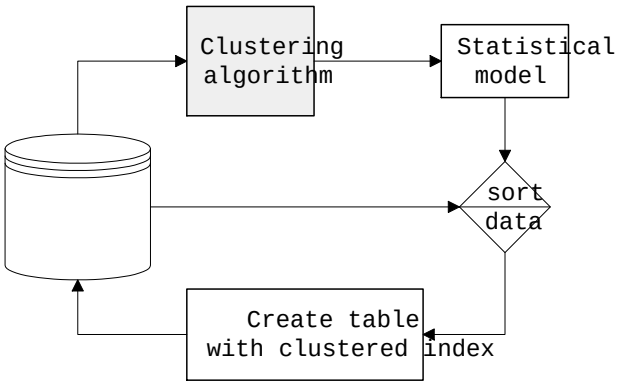

Figure 1: Overview of derivation of index structure

finding nearest neighbors at query time. The basic intuition is that there is a strong correspondence between cluster membership and proximity according to the distance measure. In this paper we develop a theory and empirical evidence to justify this intuition.

The algorithm we propose consists of two components. The first component is a pre-processor used off-line. It takes a data table, runs a statistical clustering algorithm on it, and produces a new copy of the table with a new additional column containing the cluster label for each record. This process is illustrated in Figure 1. If this new table is to be used within a database, then we assume a clustered index will be created on the new column. If it is to be used in a file system, we would likely write several files, one for each value of the new column. The second component is the query processor that scans clusters most likely to contain the nearest neighbor and terminates search when the p-approximate nearest neighbor is found.

The basic outline of our method, called DBIN (for Density-Based Indexing), is as follows:

- **Off-line:**
  1. Build a statistical estimate of the density of the data.
  2. Use the estimated density model to derive a single multi-dimensional index column on the data.
  3. Rewrite the data table so it is clustered on the new index column.
- **Query time:**
  1. Map a query to relevant values of the index column and produce an ordering by which the data pages should be scanned.
  2. As data is scanned, the statistical model is used to update a probability that the set of nearest neighbors seen so far is indeed the set of nearest neighbors.
  3. If a threshold certainty level that the solution has been found is attained, the scan is stopped.

The detailed algorithm and the analysis to support it are given in Section 4. It is important to note that at query time we require one additional query lookup stage to determine the cluster to be scanned next.

The lookup consists of using the model to determine which cluster to scan next for the given query item. The clustered index is then used to scan the members of that cluster. In a file system implementation we simply scan the file containing all the data items that belong to the cluster under consideration.

1.2 Contributions of this Paper

This paper introduces a new approach to building a multi-dimensional index structure that supports approximate nearest-neighbor queries. We introduce the notion of probabilistic-approximate nearest neighbor queries based on statistical models of the data. We model the statistical properties of the data by clustering it to produce a probability model of the data. A crucial attribute of DBIN is that the clustering algorithm utilized takes into account *all* dimensions simultaneously [10, 9]. This is essential for effective multi-dimensional indexing. Using the probability model, we produce a global ordering on the data records that maximizes proximity of records that are "near" each other according to a family of distance measures. At query time, the model allows us to use probability estimates to prioritize the search and rule out regions highly unlikely to contain points of interest. The method produces an estimate of the K nearest neighbors and a measure of confidence of the accuracy of that estimate. We demonstrate both theoretically and empirically that the scheme scales to high dimensions.

Much of the existing database literature has analyzed this problem under assumptions that data is uniformly distributed or more generally identical and independently distributed (iid) in each dimension. In this paper we adopt a more realistic assumption that the data is generated from a mixture model (a model with multiple components or clusters). Under this assumption, we derive *stability conditions* that ensure our indexing method will perform optimally. In the event that these stability conditions are violated the performance of the method degrades slowly and is still more efficient than a full scan. The statistical model we utilize also allows us to detect when our indexing structure is unlikely to be useful. Hence an optimizer may use this information to decide the tradeoff between a sequential scan and using the index structure. We present empirical results that verify our methodology and demonstrate that our model approximates real-world data distributions reasonably well. The result is an efficient technique for indexing databases for multi-dimensional nearest-neighbor queries. The method supports a general family of distance functions that includes the most commonly used ones.

2 Related Work on Indexing Methods

Much work has been done on indexing methods for nearest-neighbor and similarity queries. The most popular and successful methods are tree-based. Each tuple is represented as a point in an n-dimensional space and some strategy is used to group the data by proximity. Each group is characterized by some bounding object. The bounding objects are organized in a tree structure to support efficient querying. The type of bounding object and the method of constructing and maintaining the trees vary widely. Minimum bounding rectangles are used in many approaches, for example K-D-B-Trees, R-Trees, R*-Trees, X-trees,

and Vornoi-based indexing [25, 20, 22, 5, 3]. Pyramid-Tree indexing uses more general bounding polyhedrons [3, 1]. SS-Trees use bounding hyperspheres [31]. SR-Trees use both bounding hyperspheres and bounding boxes [23]. Seidl and Krieger have found optimal methods for calculating the optimal multi-step calculation of nearest neighbors when a lower bounding object distance is available [27]. The great value of the bounding objects is that they provide such lower bounds. Consequently, points or regions of the dataspace can be eliminated as possible candidate nearest neighbors. In this work our data is partitioned by piecewise quadratic surfaces, but these surfaces are never explicitly calculated.

These tree-based indexing methods have proven to be useful for low and moderate dimensional problems. A common strategy to take advantage of fast lower-dimensional algorithms is to map the higher-dimensional data points to a lower-dimensional space and perform the nearest-neighbor search in that space [15]. Sensible strategies for reducing dimensionality can frequently be defined based on the specific application, for example Fourier transforms in image analysis. Locality-Sensitive Hashing [19, 21] provides a general scheme for mapping data into a lower-dimensional space and finding nearest neighbors in the reduced space that are approximate ϵ nearest neighbors in the original space. DBIN also finds approximate NN but in a probabilistic sense.

When applied to high-dimensional problems, the performance of these tree-based indexes degrades markedly to the point where frequently a sequential scan is faster. This is the so called "Curse of Dimensionality". The tutorial by Berchtold and Keim [4] provides excellent insight into this phenomenon. The arguments in this section are largely based on this tutorial. For an index to be successful for nearest-neighbor queries it must accomplish two tasks. The first is to efficiently identify the nearest neighbors. The second is to terminate the search once the neighbors have been found.

An obvious strategy for finding the nearest neighbors is to attempt to group records that are proximal into the same data page. Thus the page containing a point would contain its nearest neighbor. However, most techniques ultimately determine their bounding regions by splitting on one dimension. In high dimensions this one-dimensional view is not a very effective strategy for finding clusters of data. This phenomenon is well-studied in the statistics literature [26, 29]. One-dimensional projections, especially if done in the original data coordinates of high dimensional spaces, are unreliable for determining clusters. Dimension selection strategies such as those used in TV-trees can reduce but not eliminate this problem [14], similarly for [28]. In contrast, the method we introduce is based on clustering algorithms such as EM [10]. Clustering uses all the dimensions simultaneously and addresses this problem directly.

In traditional indexing methods, confirming that the nearest neighbor has been found presents the most difficult task. Usually a bounding object is used and if the estimated query ball intersects the region defined by the object the corresponding path must be examined. Much of the problems stem from the fact that volumes of these bounding objects grow exponentially with dimension. Thus selectivity of queries grows rapidly with dimension and data pages have large extent. In [2] an approximate NN-model based on assumption of uniform distribution of the data found that in high dimensions a full scan will be required. A key point about this work is that it assumes that if a nearest-neighbor query ball intersects a data page

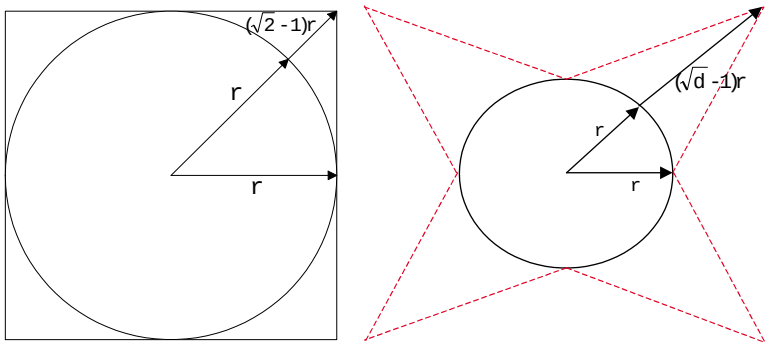

Figure 2: As d goes to infinity, the corners hold all the volume.

the data page must be visited. Thus if a NN-query ball intersects most of the data pages then the entire data set must be scanned. This is the downfall of traditional indexing schemes in high-dimensions. Using the traditional techniques of lower bounding the distance between the query point and the objects in a data page does not eliminate the page from consideration. Therefore the algorithm must visit the pages even though there is a very low probability that the nearest neighbor actually resides within the page.

The answer to this "curse of dimensionality" is not tighter bounds through better partitioning of the dataspace nor different choices of bounding objects. Consider an example adapted from Berchtold and Kiem's tutorial [4]. Suppose a set of data points resides densely in an d-dimensional sphere of radius r and that we construct a minimum bounding rectangle about this sphere. The two-dimensional case is shown in Figure 2 (left). Note the ratio of the volume of the inscribed sphere to the volume of the minimum bounding box converges rapidly to 0 as d goes to infinity. Specifically, $\left[\frac{r^d\sqrt{\Pi^d}}{\Gamma(d/2+1)}\right]\left[\frac{1}{(2r)^d}\right] \rightarrow 0$. Define the "corners" of the bounding box as the regions outside the sphere but still in the box. When d is very large, most of the volume is in the corner of the boxes and yet there is no data in the corners. Figure 2 (right) suggests this growth of the corners – in reality both the number and size of the corners grows exponentially with dimensionality.

Any fixed geometry will never fit real data tightly. In high dimensions, such discrepancies between volumes and density of data points start to dominate. Increase in overlap with dimensionality such as reported in [5] is a practical manifestation of this phenomena. Even in the best case, where the bounding object is the convex hull of the points and objects within it, the data is still very sparse and not necessarily uniformly distributed. So large regions within the convex hull may contain no points. Indices based on bounding objects are not good enough, because they frequently scan data pages where no relevant data resides. Our probabilistic approach allows us to identify data partitions most likely to contain relevant data and eliminate regions unlikely to contain relevant points. The price we pay is that we can only provide estimates of the nearest neighbor with a user-specified confidence rather than with certainty.

Other researchers have also found that using using ϵ-approximate nearest-neighbor queries can avoid the "curse of dimensionality" (see [21] for overview). These methods are guaranteed to find points no more that $1+\epsilon$ farther away than the true nearest neighbor. Theoretically these algorithms can have sublinear query

time but frequently the query time depends inversely on ϵ. One of the most practical of these methods is locality sensitive hashing. It's interesting to note that locality-sensitive hashing is a randomized algorithm that guarantees finding ϵ-approximate nearest-neighbor queries with constant probability. So in a weaker since it is also a probabilistic nearest-neighbor algorithm.

Another work of interest is that of Beyer et al. [6]. They question the validity of nearest-neighbor queries in high dimensions. Under commonly used assumptions such as data drawn from a distribution that is independent and identically distributed in each dimension, the ratio of the nearest-neighbor and the farthest-neighbor distance approaches one as the dimensionality increases. So, for example, if data is drawn from a single Gaussian in very high dimensions, then all the points are roughly the same distance apart. Beyer et al state that these assumptions are violated in clustered data. Herein we extend their results to show that for clustered data generated from a mixture model of well-separated Gaussians, nearest neighbor is meaningful and efficient indexing methods exist based on clustering the data and modeling the contents statistically.

3 Modeling the Data with a Density Estimate

A notion central to our approach is that of estimating the density of the data. By density we mean a joint probability density function f governing the distribution of data values. We can take several approaches to density estimation [26]. A major consideration, however, is the efficiency of the method and the suitability of the statistical model to the task at hand: the nearest-neighbor problem. For these reasons, we chose to model the data using a mixture of Gaussians. Hence the pdf has the form:

$$f(x) = \sum_{i=1}^{K} p_i G(x|\mu_i, \Sigma_i)$$

where $G(x|\mu_i, \Sigma_i)$ is a Gaussian pdf parameterized with a mean vector μ_i and a covariance matrix Σ_i. There are several desirable properties of the Gaussian mixture model for this problem including:

1. Expressive power: any distribution can be represented as a mixture of Gaussians [26, 29].
2. Computational efficiency: using our existing scalable approaches to estimate the density in this form [10], as we discuss in Section 3.1.
3. Suitability to distance measure: as we show in Section 4, we can exploit this form to help construct index structures for a large family of distance functions.
4. Ability to analyze the model, study its properties, and convenience of its use to compute probabilities of regions and events of interest.

3.1 How do we obtain the desired density estimate?

We use data clustering (a.k.a. segmentation) algorithms as an efficient means of obtaining the statistical model. In clustering, the input is a number of clusters, an initial guess at where the clusters are centered,

and what their initial parameters (covariances) are. The initial guess is typically some random setting unless one has prior knowledge of the data. For more details on initialization of clustering algorithms see [8, 17]. A clustering algorithm such as K-Means [9, 18, 32] or EM [10, 7, 12] simply iterates over the data and maps the initial parameters to a new set of parameters that result in a statistical model which fits the data better. The traditional K-Means algorithm assumes all clusters are modeled by Gaussians that have identical, diagonal covariance matrices with a constant entry on all diagonals. Thus it models data by spherical Gaussians. In this simplified setting, a data point simply belongs to the cluster whose center is nearest to the point (under Euclidean distance). K-Means is an iterative algorithm and is guaranteed to converge to a local minimum. Its complexity is linear with the size of the data. Since we do not want to restrict ourselves to spherical Gaussians, we use a more general algorithm called EM (for Expectation Maximization) [10, 12, 7].

EM iterates in exactly the same way as K-means, except that it uses a *soft* or *probabilistic* membership assignment rule. Each data point belongs to each cluster, but with a weight determined by the Gaussian used to represent each cluster. We use a scalable implementation of EM [10] as a tool to obtain the desired clustering. The algorithm is discussed in detail in [10] and can build models over large databases accurately and in a single scan. In this paper we assume that the clustering model is given as an input. We focus on its use to derive a multidimensional index.

3.2 Using the Statistical Model

The model consisting of a mixture of K Gaussians determined by scalable EM [10] is used in two important ways:

1. determining how the data table should be sorted and materialized on disk with a new clustered index.
2. as part of the indexing scheme at query time, the model plays two important roles:
   (a) Determine the order in which, for a given query, the data clusters should be scanned.
   (b) During the data scan, determine the probability that the target nearest neighbors have been encountered and hence when it is safe to refrain from scanning other clusters.

Determining the membership of a data point $x \in D$ in a cluster $C_j \in \{C_1, \ldots, C_K\}$ is achieved by finding the probability of x under each of the clusters (Gaussians) and assigning it to the cluster with the highest probability:

$$\mathrm{P}\ (x|C_j) = \frac{G(x|\mu_j, \Sigma_j)}{\sum_{i=1}^{K} G(x|\mu_i, \Sigma_i)}$$

where $G(x|\mu_i, \Sigma_i)$ is the Gaussian representing cluster C_i. Recall that a given Gaussian G with dimension d, mean vector μ and covariance matrix Σ is given by the standard form:

$$\mathrm{G}(x|\mu, \Sigma) = \frac{1}{\sqrt{2\pi^d |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

The next step is to materialize the table on disk sorted by cluster membership. Alternatively, data from each cluster can be written in a separate file if the scheme is to be used with a file system rather than in a

DBMS. At query time, DBIN first decides which cluster to scan first by finding the most probable cluster given the query. It scans the data in that cluster and keeps track of the nearest-neighbor distance (or K nearest neighbors). Once it finishes the first cluster scan, DBIN needs to determine whether other clusters (whose centroids are by definition further than clusters scanned so far) need to be scanned. To do this, we determine a region of interest in the space that we call the *query ball*. The query ball is centered at the query point and has as its radius the distance to the nearest neighbor (or farthest of nearest neighbors) found so far. We now need to determine the probability of this query ball region under each of the remaining clusters. This involves computing the probability of a region under a Gaussian and is obtainable using several approximations. The details are covered in Section 4. This determination gives us the next cluster to be scanned. In addition, it results in an estimate of the probability that *we are done* (i.e. that we have found the nearest neighbors already). We use this probability as a basis for our stopping criterion.

3.3 Does It Work in High Dimensions?

The primary distinguishing characteristic of DBIN is its use of a model of the density to reorganize the data. As discussed in Section 2, previous indexing schemes have not had much success in scaling to high dimensions. This is explained by two factors. The first is that in any method high dimensions are harder to work with. The second factor is that most indexing schemes, due to efficiency considerations, define regions by examining one dimension at a time. Projections to a single dimension can be very confusing when the data is high-dimensional. An indexing method that constructs a tree based on one dimension at a time, is likely to fail to detect basic multidimensional structure in the data such as clusters and gaps. In contrast, clustering works on all dimensions simultaneously to discover this structure. In the past, clustering has not been convenient or efficient for large data sets. However, with recent data mining advances, this has changed and we are now able to cluster large databases effectively and efficiently.

As discussed in Section 2, in high-dimensional spaces, a query region can easily overlap with most index regions. We overcome the problem by providing a mechanism for *prioritizing* regions (clusters in our case), and hopefully *rule out* a subset of them as highly unlikely to contain data records relevant to the query. This contrasts with traditional indexing schemes which visit every data region intersecting the query region since the decision as to whether a region overlaps with a point is binary rather than probabilistic.

Finally, most analyses of problems with indexing schemes assume that the data is independent and identically distributed in each dimension (e.g. the data is from a uniform or single spherical Gaussian distribution [6, 2, 3]). While in such cases it is not clear that nearest-neighbor makes sense at all (see [6] for a good exposition, also discussed in Section 2) we demonstrate in this paper that if the data is clustered, then indeed nearest-neighbor queries are meaningful and it is possible to construct a useful indexing scheme. This leads us to definitions of the notion of *data stability* from a nearest-neighbor perspective (Section 5). In well-separated clusters, the nearest neighbor of a point is always in the same cluster as that point, regardless of the dimensionality.

4 The Algorithm and Its Probabilistic Analysis

In this section we describe our algorithm (DBIN) for answering nearest-neighbor queries in sufficient detail to permit analysis. DBIN has two main components. The first component takes a data set D as input and constructs an index that supports nearest-neighbor queries. The second component takes a query point q and produces the nearest neighbor of q with high probability.

The index is constructed in three steps.

1. Produce a best-fit mixture-of-Gaussians probability density function (pdf) for the data;
2. Use the optimal Bayes decision rule to assign each data point to a cluster;
3. Sort the data points by their cluster assignment.

There are many ways to find a mixture-of-Gaussians pdf that fits data (e.g., Thiesson et al., 1998, [30]). We use a scalable EM algorithm [10] that was developed to classify large databases for a variety of applications other than nearest-neighbor queries. The outcome of this algorithm is a mixture-of-Gaussians pdf of the form:

$$f(x) = \sum_{i=1}^{K} p_i G(x|\mu_i, \Sigma_i) \tag{1}$$

where p_i are the mixture coefficients, $0 < p_i < 1$, $\sum p_i = 1$, and $G(x|\mu_i, \Sigma_i)$ is a multivariate Gaussian pdf with a mean vector μ_i and a positive definite covariance matrix Σ_i. The optimal decision rule for a mixture-of-Gaussians pdf is presented for example in [13, Chapter 2]. It dictates that a data point x is assigned to cluster C_l if $i = l$ maximizes the quantity

$$g_i = -1/2(x - \mu_i)^T \Sigma^{-1}(x - \mu_i) - 1/2 \log|\Sigma_i| + \log p_i. \tag{2}$$

Note that this decision rule maximizes the probability that x is generated by the i^{th} Gaussian distribution in Equation 1 (cluster C_i) and in that sense it is an optimal decision. This decision rule defines a set of K regions $R_1, \ldots, R_k$ in high-dimensional Euclidean space where R_i contains the set of points assigned to cluster C_i. We define $R(x)$ to be the *region-identification function*; that is, each data point x belongs to region $R(x)$ where $R(x)$ is one of the K regions $R_1, \ldots, R_k$. Finally, an index $I(x)$ is defined such that $I(x) = i$ if $R(x) = R_i$, namely, I has K possible values which indicate to which cluster the optimal Bayes rule assigns x. The sorting of the database by $I(x)$ can be implemented by adding a new attribute to the database and by building a clustered index on it.

The second component of DBIN is activated at query time. It finds with high probability a nearest neighbor of a query point q, denoted by $n(q)$. For its description we use the notation $B(q, r)$ to denote a sphere centered on q and having radius r. We also denote by E the set of regions scanned so far, and by e the knowledge learned by scanning the regions in E. A nearest neighbor of q is then found as follows.

NearestNeighbor (q ; $R_1, \ldots, R_k, f$)

1. Let C_j be the cluster assigned to q using the optimal decision rule (Equation 2);

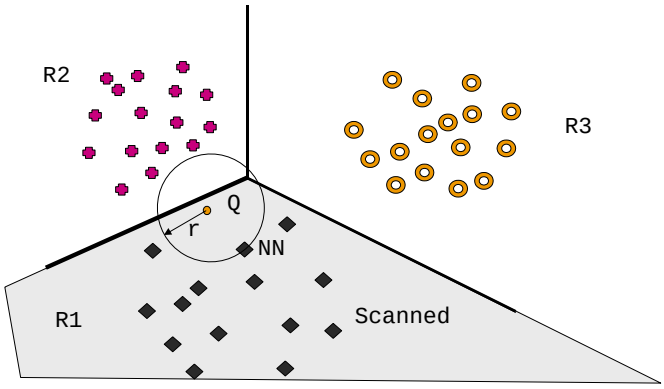

Figure 3: Nearest neighbor query in stable database

2. Scan data in R_j and determine nearest neighbor $n(q)$ in R_j and its distance r from q;
3. Set $E = \{R_j\}$;
4. While $\mathrm{P}(B(q,r)$ *is Empty* $\mid e) <$ *tolerance*
   - **a.** Find a cluster C_j not in E which minimizes $\mathrm{P}(B(q,r) \cap R_j$ *is Empty* $\mid e)$;
   - **b.** Scan the data in R_j; Set $E = E \cup \{R_j\}$;
   - **c.** If a data point closer to q is found in R_j, let $n(q)$ be that point, and set r to be the new minimum distance.

The quantity $\mathrm{P}(B(q,r)$ *is Empty* $\mid e)$ is the probability that $B(q,r)$, which we call the query ball, is empty, given the evidence e collected so far. The evidence consists simply of the list of points included in the regions scanned so far. Before we show how to compute this quantity and how to compute $\mathrm{P}(B(q,r) \cap R_i$ *is Empty* $\mid e)$, we explain the algorithm using the simple example depicted in Figure 3. In this example, the optimal Bayes rule generated three regions R_1, R_2 and R_3 whose boundaries are shown. A given query point q is found to reside in region R_1. The algorithm scans R_1, a current nearest neighbor is found, a current minimum distance r is determined, and a query ball $B(q,r)$ is formed. The query ball is shown in the figure. If our algorithm was deterministic it would be forced to scan R2 and R3 since they intersect the query ball. Instead the algorithm determines the probability that the ball is empty given the fact that region R_1 has been scanned. Suppose this probability does not exceed the tolerance. The algorithm must now choose between scanning R_2 and scanning R_3. A choice should be made according to the region that maximizes the probability to find a nearer neighbor once that region is scanned, namely, the algorithm should scan the region that minimizes $\mathrm{P}(B(q,r) \cap R_i$ *is Empty* $\mid e)$. This quantity is hard to compute and so the algorithm approximates this quantity using Equation 5. Below we show how to compute the approximation and analyze the difference between the computed quantity and the desired one. In this example, region R_2 is selected to be scanned. The algorithm now halts because $\mathrm{P}(B(q,r)$ *is Empty* $\mid e)$ becomes negligible once R_1 and R_2 have been scanned.

The basis for computing $\mathrm{P}(B(q,r)$ *is Empty* $\mid e)$ is the fact that data points are assumed to be randomly generated using the mixture-of-Gaussians pdf f (Equation 1). In other words, we take f to be the true

model that generated the database. The sensitivity of the algorithm to this assumption must be tested using real data sets. The probability the ball is empty is the product of the probabilities that each point does not fall in the ball, because, according to our assumption, the x_i's are iid samples from $f(x)$.

Consequently, we have

$$P(B(q,r) \textit{is empty} \mid e) = \prod_{i=1}^{n} [1 - P(x_i \in B(q,r) \mid x_i \in R(x_i))] \tag{3}$$

where $R(x_i)$ is the region of x_i and n is the number of of data points. If $R(x_i)$ has been scanned then $P(x_i \in B(q,r) \mid x_i \in R(x_i)) = 0$. In general, $P(x_i \in B(q,r) \mid x_i \in R(x_i))$ is not computable. Fortunately, Theorems 4.1 and 4.2 below show that we can use the approximation

$$P(x_i \in B(q,r) \mid x_i \in R(x_i)) \approx P(x_i \in B(q,r) \mid x_i \text{ generated by } C_j) \tag{4}$$

where C_j is the cluster assigned to x_i using the optimal Bayes decision rule.

Use the same reasoning as above and the fact $P(x_i \in B(q,r) \cap R_j | x_i \in R_j) = P(x_i \in B(q,r) | x_i \in R_j)$,

$$\begin{aligned} \mathrm{P}(B(q,r) \cap R_j \textit{ is Empty} \mid e) &= [1 - P(x_i \in B(q,r) \mid x_i \in R_j)]^{n(R_j)} \\ &\approx [1 - P(x_i \in B(q,r) \mid x_i \text{ generated by } C_j)]^{n(R_j)} \end{aligned} \tag{5}$$

where $n(R_j)$ is the number of points falling in region R_j.

The remaining task of computing $P(x \in B(q,r) \mid x \text{ generated by } C_l)$ has been dealt with in the statistical literature in a more general setting. This probability can be calculated numerically using a variety of approaches [24]. In particular, numerical approaches have been devised to calculate probabilities of the form

$$P[(x-q)^T D (x-q) \leq r^2] \tag{6}$$

where x is a data point assumed to be generated by a multivariate Gaussian distribution $G(x|\mu, \Sigma)$ and D is a positive semi-definite matrix. If the Euclidean distance is used to measure distances between data points, as we do herein, D is simply the identity matrix. However, the numerical methods apply to any distance function of the form $d(x,q) = (x-q)^T D (x-q)$ where D is a positive semi-definite matrix.

The pdf of the random variable $(x-q)^T D (x-q)$ is a χ^2 distribution when $D = I$ and $q = \mu$. It is a noncentralized χ^2 when $D = I$ and $q \neq \mu$. It is a sum of non-centralized χ^2 pdfs in the general case of a positive semi-definite quadratic form e.g., $x^T D x \geq 0$ for any x. The general method uses a linear transformation to reduce the problem to calculating the cumulative distribution of a linear combination of chi-squares. We use the method of Sheil and O'Murcheartaigh [24].

We illustrate the idea of the computation assuming that the dimensions are uncorrelated (i.e. Σ is diagonal) and that the Euclidean distance metric is used (D is the identity matrix), but we emphasize again that the needed probability is computable for the general case. We start with

$$(x-q)^T D (x-q) = \sum_{j=1}^{n} (x_j - q_j)^2, \tag{7}$$

where q_j is the j-th component of q and transform x_j to a standard normal random variable z_j.

$$(x_j - q_j)^2 = \sigma_j^2 \frac{((x_j - \mu_j) + (\mu_j - q_j))^2}{\sigma_j^2} = \sigma_j^2 (z_j + \delta_j)^2 \tag{8}$$

where $\delta_j = \frac{(q_j - \mu_j)}{\sigma_j}$ and μ_j is the j-th component of μ. Now $(z + \delta_j)^2$ has a noncentral chi-square distribution with 1 degree of freedom and noncentrality parameter δ_j^2. We denote this as $\chi^2(1, \delta_j^2)$. The final result is

$$P[(x-q)^T D(x-q) \leq r^2] = P\left[\sum_{j=1}^{n} \sigma_j^2 \chi^2(1, \delta_j^2) \leq r^2\right] \tag{9}$$

This results assumes $\sigma_j \neq 0$ for all j. In practice some dimensions may have zero variance, i.e. $\sigma_j = 0$. In this case

$$P[(x-q)^T D(x-q) \leq r^2] = P\left[\sum_{\sigma_j \neq 0} \sigma_j^2 \chi^2(1, \delta_j^2) \leq r^2 - \sum_{\sigma_j = 0} (q_j - \mu_j)^2\right] \tag{10}$$

This cumulative distribution function (cdf) can be expanded into an infinite series. In the algorithm, we used, AS 204 [16], the terms of the series are calculated until an acceptable bound on the truncation error is achieved. Since we are estimating small probabilities with very high accuracy, this may not be the most appropriate algorithm. It achieved adequate results in our experiments but suffers from numerical accuracy problems on high-dimensional instances with large noncentrality parameters. Several other techniques for calculating or estimating quadratic normal forms have been proposed [24] that They may be more numerically stable for high-dimensional problems. We are in the process of testing their suitability for this problem.

We have examined theoretically the accuracy of the approximation used in the probability calculation (4). The first theorem shows that what we compute with these known techniques is not very far from what we would have desired to compute. Let B be the event "*$B(q,r)$ is Empty*," C_1 be the event "*x is generated by C_1*," and R_i be the event "$x \in R_i$".

Theorem 4.1 *Let the events B, R_1, and C_1 be as defined above. Then,*

$$|P(B|R_1) - P(B|C_1)| \leq P(B)(1/P(C_1) + 1/P(R_1))$$

Proof. The inequalities $P(B|R_1) \leq P(B)/P(R_1)$ and $P(B|C_1) \leq P(B)/P(C_1)$ imply the claim. □
Note that the bound gets arbitrarily tight as the sample size increases because the radius of a ball around q converges to zero as the sample size increases. Another error bound is:

Theorem 4.2 *Let the events B, R_1, and C_1 be as defined above. Then,*

$$|P(B|R_1) - P(B|C_1)| \leq P(B \cap R_1 \cap C_1) \left[\left|\frac{1}{P(R_1)} - \frac{1}{P(C_1)}\right| + P(B)\left|\frac{1}{P(C_1 \cap B)} - \frac{1}{P(R_1 \cap B)}\right|\right]$$

Proof. Using the definition of conditional probability and the triangle inequality, we have

$$
\begin{aligned}
&|P(B|R_1) - P(B|C_1)| \\
&= |\tfrac{P(B\cap R_1\cap C_1)}{P(R_1)} + \tfrac{P(B\cap R_1\cap \bar{C}_1)}{P(R_1)} - \tfrac{P(B\cap R_1\cap C_1)}{P(C_1)} - \tfrac{P(B\cap \bar{R}_1\cap C_1)}{P(R_1)} \\
&= |P(B\cap R_1\cap C_1)(\tfrac{1}{P(R_1)} - \tfrac{1}{P(C_1)}) + P(B)(P(\bar{C}_1|R_1\cap B) - P(\bar{R}_1|C_1\cap B))| \\
&= |P(B\cap R_1\cap C_1)(\tfrac{1}{P(R_1)} - \tfrac{1}{P(C_1)}) + P(B)(P(R_1|C_1\cap B) - P(C_1|R_1\cap B))| \\
&= |P(B\cap R_1\cap C_1)(\tfrac{1}{P(R_1)} - \tfrac{1}{P(C_1)}) + P(B)P(B\cap R_1\cap C_1)(\tfrac{1}{P(C_1\cap B)} - \tfrac{1}{P(R_1\cap B)})| \\
&\leq P(B\cap R_1\cap C_1)\left[|\tfrac{1}{P(R_1)} - \tfrac{1}{P(C_1)}| + P(B)|\tfrac{1}{P(C_1\cap B)} - \tfrac{1}{P(R_1\cap B)}|\right].\square
\end{aligned}
$$

This shows the the smaller the probability that the ball intersects the region then the better the bound. Note that the Bayes optimal decision rule minimizes the differences between $P(C_i)$ and $P(R_i)$ which means that this bound for a typical pair R_i, C_i is on the average as tight as possible when the optimal Bayes decision rule is used to cluster the data points.

5 Nearest-Neighbor Behavior within Gaussian Mixtures

In previous sections we showed that the Gaussian model could be used to prioritize search and predict the accuracy of an estimated nearest neighbor. In this section we examine under what conditions DBIN should perform well. Under the assumptions of a mixture of Gaussians, we define stability conditions that ensure that nearest-neighbor queries are meaningful and that the nearest neighbor of a point is in the same cluster as the point. If the nearest neighbor is always in the same cluster, then DBIN need only scan one cluster. Note that all of these results are asymptotic in the sense that they hold as the attribute dimensionality increases to infinity. Full proofs are given in the appendix.

Our results extend those of Beyer et al [6]. They proved the disturbing result that nearest-neighbor queries were meaningless in high dimensions under commonly used assumptions of data distributions. The difficulty is that the ratio of nearest and farthest neighbor distance converges in probability to 1 as the dimensionality increases. We adopt the notation of [6].

Definition 5.1 (Notation)

d (usually interpreted as dimension of space)

$F_1, F_2, \ldots$ is a sequence of data distributions.

$Q_1, Q_2, \ldots$ is a sequence of query distributions.

n is fixed number of samples from each distributions.

For any d, $x_{d,1}, x_{d,2}, \ldots, x_{d,n}$ are n independent data points per d such that $x_{d,i} \sim F_d$.

q_d is a query point generated independently from Q_d.

We will use the squared Euclidean distance, i.e. $||x_{d,i} - q_d||^2$, however, this work is generalizable to other distance measures.

$DMIN_d = min\{||x_d - q_d||^2 | 1 \leq i \leq n\}$.

$DMAX_d = max\{||x_d - q_d||^2 | 1 \leq i \leq n\}$.

Beyer et al's result in this notation is:

Theorem 5.1 (Meaningless Nearest Neighbor Queries [6]) *If*

$$\lim_{d\to\infty} var\left(\frac{\|x_{d,1}-q_d\|^2}{E[\|x_{d,1}-q_d\|^2]^2}\right) = 0 \tag{11}$$

Then for every $\epsilon > 0$

$$\lim_{d\to\infty} P[DMAX_d \leq (1+\epsilon)DMIN_d] = 1. \tag{12}$$

Theorem 5.1 applies to data generated by a single Gaussian. In a mixture of Gaussians model, each data point or query is generated by a single Gaussian. We can think of the points as being clustered by the Gaussian that generated them. We assume that the data point and the query points are generated by the same distribution. In the proof of Theorem 5.1, Beyer et al showed that the distance between points in the same cluster approaches the mean *within cluster* distance as the dimensionality increases. We say that the *within cluster* distance is **unstable** because roughly every point is the same distance apart. This result is precisely stated as follows:

Corollary 5.1 (*Within Cluster* **Distances Converge)** *If*

$$\lim_{d\to\infty} \frac{var(\|x_{d,1}-q_d\|^2)}{E(\|x_{d,1}-q_d\|^2)^2} = 0 \tag{13}$$

Then

$$\frac{\|x_{d,1}-q_d\|^2}{E(\|x_{d,1}-q_d\|^2)} \longrightarrow_p 1. \tag{14}$$

Similarly, the distance between any two points from two distinct clusters approaches the mean *between cluster* distance.

Corollary 5.2 (*Between Cluster* **Distances Converge)** *If*

$$\lim_{d\to\infty} \frac{var(\|x_{d,2}-q_d\|^2)}{E(\|x_{d,1}-q_d\|^2)^2} = 0 \quad and \quad \lim_{d\to\infty} \frac{E(\|x_{d,2}-q_d\|^2)}{E(\|x_{d,1}-q_d\|^2)^2} = \delta. \tag{15}$$

Then

$$\frac{\|x_{d,2}-q_d\|^2}{E(\|x_{d,1}-q_d\|^2)} \longrightarrow_p \delta. \tag{16}$$

If for two clusters, the *between cluster* distance dominates the *within cluster* distance, we say the clusters are **stable** with respect to each other. Figure 4 gives examples of clusters that are **stable** and **unstable**.

Theorem 5.2 (Pairwise *Between Cluster* **Stability)** *Assume* q_d *and* $x_{d,i}$ *are generated by cluster* i *and* $x_{d,j}$ *is generated by cluster* j*. If*

$$\frac{\|x_{d,i}-q_d\|^2}{E(\|x_{d,i}-q_d\|^2)} \longrightarrow_p 1 \quad and \quad \frac{\|x_{d,j}-q_d\|^2}{E(\|x_{d,i}-q_d\|^2)} \longrightarrow_p \delta > 1. \tag{17}$$

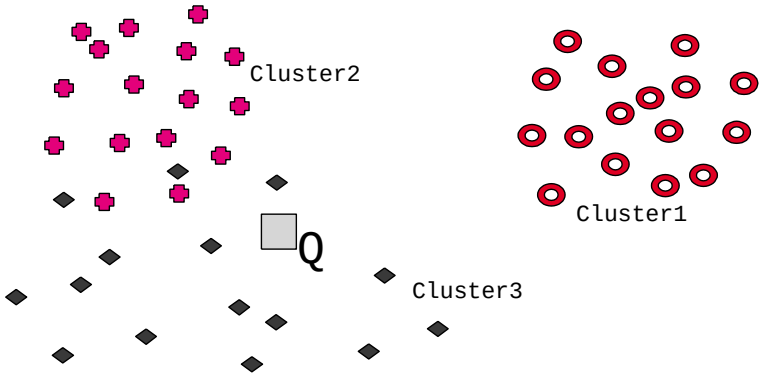

Figure 4: Cluster 1 is stable with respect to 2 and 3. Clusters 2 and 3 are not stable.

Then we say Clusters i and j are pairwise stable with parameter δ and for any $\epsilon > 0$

$$\lim_{d\to\infty} P(\|x_{d,j} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2) = 1 \tag{18}$$

If every cluster is **stable** with respect to at least one other cluster then nearest neighbor is well defined in the sense that the nearest and farthest neighbor distances are bounded apart. With probability 1, the ratio of the farthest and nearest neighbors is bigger than some constant greater than 1. For example in Figure 4, Cluster 1 is stable with respect to both Clusters 2 and 3 so nearest neighbor is well defined.

Theorem 5.3 (Nearest Neighbor Well-Defined) *Assume q_d and $x_{d,i}$ are generated by cluster i and $x_{d,j}$ is generated by cluster j. If cluster i is unstable with itself and there exists a cluster j that is pairwise stable with i with parameter $\delta > 1$, then for any $\epsilon > 0$*

$$\lim_{d\to\infty} P[DMAX_d \geq (\delta - \epsilon)DMIN_d] = 1 \tag{19}$$

If every cluster is **stable** with respect to every other cluster then if a point belongs to one cluster, its nearest neighbor also belongs to that cluster. Therefore if we partition our data by cluster membership then with probability one our index will only need to visit one cluster to find the nearest neighbor. With probability one, other clusters can be skipped with no false drops of points.

Theorem 5.4 (Nearest Neighbor in Cluster) *Assume q_d and $x_{d,i}$ are generated by cluster i. If cluster i is unstable with itself and and pairwise $\delta_{ij} > \delta > 1$ stable with every other cluster j, $j \neq i$, then q_d nearest neighbor was also generated by cluster i. Specifically for any point x_{d_j} from cluster $j \neq i$*

$$P[\|x_{d,j} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2] = 1. \tag{20}$$

These results show that if we have a stable mixture of Gaussians where the between cluster distance dominates the within cluster distance, then if we partition by a cluster membership function that assigns all data generated by the same Gaussian to the same partition, the index would work perfectly for nearest-neighbor queries generated by the same distribution. The higher the dimensionality, the better it would

work. There is no "curse of dimensionality" in this case. In Appendix B we give an example of the stability rule derived for a mixture of spherical Gaussian with identical covariance matrices.

Note that we actually do not know which Gaussian generates a data point. But, as discussed in Section 4, we can use the optimal Bayes decision rule [13] to estimate the Gaussian that generated a point with minimum error. Figure 3 illustrates stable clusters and the corresponding partitioning of the data space by the optimal Bayes rule.

6 Computational Results

Our initial experimental goals were to confirm the validity of the theoretical results, to determine the accuracy of the probability estimates, and to establish that DBIN would be practical on real-world data with unknown distribution. We assumed that if a cluster is visited that the entire cluster is scanned. We did not address paging of data within a cluster. We experimented with both synthetic and real-world databases. The purpose of synthetic databases is to study the behavior of DBIN in well-understood situations. The experiments on the real data sets verify that our assumptions are not too restrictive and apply in natural situations.

6.1 Synthetic Databases

First we verified the stability theory introduced in Section 5 by applying DBIN to both stable and unstable mixtures of Gaussian data. We used synthetic data sets drawn from a mixture of ten Gaussians. First, we used *stable* clusters with a known generating model, then *unstable* clusters with a known generating model, and finally, *stable* clusters with an unknown generating model, i.e. a situation in which we had to estimate the density (do the clustering).

Stable Case, Known Probability Density: We generated a mixture of ten Gaussians in d dimensions with distance τ_d between the means of the distribution. Each Gaussian had covariance matrix $\sigma^2 I$ where I is the identity matrix and $\sigma^2 = .01$. As discussed in Section 5, if $\tau_d > \sigma\sqrt{d}$, the clusters are stable. In order to fix the distance between the means to be τ_d apart, we set the i^{th} mean $\mu_i = \sqrt{\tau_d/2}e_i$ where e_i is a vector of zeros with a one in the i^{th} component. The size of the database was fixed and we generated $500000/d$ points for $d < 100$ dimensions and $1000000/d$ for $d \geq 100$. DBIN was used to find the two nearest neighbors (2 NN) for 250 query points randomly selected from the database. To remove any variation due to inaccuracies in clustering, we first used the *true generating density* as input to DBIN. Hence, this represents an extreme for a best-case scenario.

The stable case, with $\tau_d = \sigma d$ was evaluated on 10, 20, 30, 40, 50, 60, 70, 100, 200, and 500 dimensions. In every case the 2 NN were found by examining only one cluster. In addition DBIN correctly determined that no additional clusters needed to be searched. Since each cluster contained ten percent of the data, each query required a scan of ten percent of the data. Similar experiments for 10 NN produced the same conclusion. In the best of worlds when the clusters are stable and the model is known, DBIN works perfectly. As the theory in Section 5 predicted, the "curse of dimensionality" vanishes when the distributions are stable.

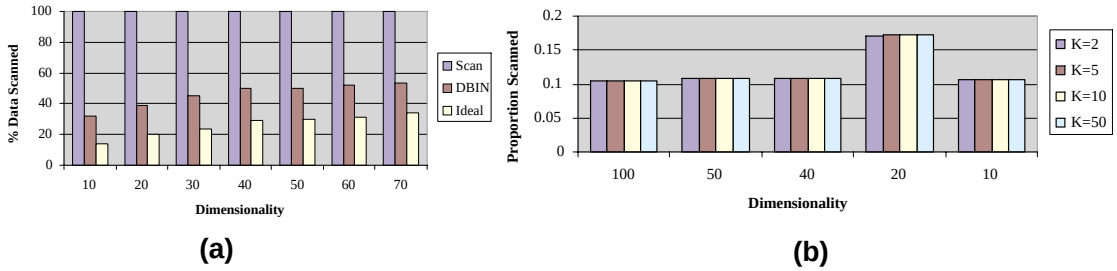

Figure 5: Fraction scanned: (a) Unstable case– $\tau_d = 2\sigma$ and (b) Unknown stable model.

Dimension	10	20	30	40	50	60	70
Accuracy	98.8	96.4	93.6	93.2	94.8	96.4	92.4

Table 1: Percentage accuracy of DBIN on unstable data

Unstable Case, Known Probability Density For unstable data, we fixed $\tau_d = 2\sigma$. With this distribution, any problem with dimensionality over 4 is unstable. The amount of overlap of the Gaussians is growing exponentially with d. This is a worst-case scenario as there is no separation between the clusters and the asymptotic stability theory does not apply.

To evaluate how well DBIN estimated when to stop scanning, we compared it with the Ideal approach that scans the clusters in the prioritized order predicted by DBIN and stops scanning additional clusters once the NN is found. DBIN stops scanning when the NN is found with high confidence. Since this case is very unstable, we would expect any algorithm to require a scan of much of the database. The percentages of data scanned by a full scan, DBIN, and the Ideal approach are given in Figure 5(a). Table 1 provides the percentage of times an error occurred, i.e. the estimated nearest neighbor distance exceeded the actual NN distance. The percentage of the data scanned increased gradually with dimensionality. The Ideal algorithm scanned less data. This difference between the Ideal algorithm and DBIN indicates that the DBIN probability estimate is conservative. In many cases DBIN slightly overestimates the probability of a NN residing in a cluster so there may be an opportunity for tightening the probability estimate found by DBIN.

Random Stable Clusters, Unknown Probability Density, Varying K: In the next set of experiments we applied the full DBIN approach to generated data. We utilized data and corresponding clusters generated for a prior paper [10] unchanged. The data were generated from 10 Gaussians with means independently and identically distributed in each dimension from a uniform distribution on $[-5, 5]$. Each diagonal element of covariance matrices was generated from a uniform distribution on $[0.7, 1.5]$. Hence this data is well-separated and should be stable, but is not at an extreme of stability. We used clusters generated by the EM algorithm without using knowledge of the known distribution except the number of clusters. Results on problems with 10 to 100 dimensions, with K (number of nearest neighbors to find) varying from 2 to 50, are given in Figure 5(b). Note that DBIN *made no errors at all in finding the nearest neighbors for all values of K.*

K	2	5	10	50
Accuracy	99.0%	100%	100%	97.1%
Fraction Scanned	12.5%	13.9%	14.2%	16.6%

Table 2: Accuracy and Fraction Scanned Results for 11-dimensional Census Data

Hence there is no point in plotting those results.

It turns out this distribution is stable in practice. This is no surprise since in high dimensions the distance between the uniformly generated means approaches the expected value which is proportional to $\sqrt{d}$. The 20-dimensional result is an artifact of the fact that our EM clustering algorithm happened to find a non-optimal solution. Hence more than 1 cluster is scanned on average. Note that because we limited ourselves to 10 clusters, 0.1 is the minimum possible scan fraction here. Thus we are near the optimal scan possible. Larger number of clusters will result in less data scanning.

6.2 Real Database Results

We experimented on real-world datasets. Overall the results are promising, but we found some additional issues to resolve. In the current implementation of DBIN, we did not address categorical data fields. These fields violate the assumption that the data has a mixture of Gaussian distribution. Some numerical stability problems occur with sparse datasets that result in near-zero variance clusters. The algorithm still performed as we expected. We need to extend the theory to the case where data is not truly continuous. Since our primary goal in this paper is to introduce the basic idea, and just demonstrate that it is possible to index higher dimensional data, the results presented here are not optimized (i.e. we did not build the best possible clustering model). Nor did we optimize the algorithm to gracefully deal with small variances or discrete data masquerading as continuous values. The goal is to demonstrate that in principle we can derive useful indexing information from statistical structure. All results are averaged over 1000 query points drawn randomly from the data.

U.S. Census Data: The publicly available U.S. Census Bureau data set consists of 300K records. We selected the 11 dimensions that appeared to be numeric. These were fields such as ages, incomes, taxes, etc. Table 2 shows the results for varying K. The model we constructed had 100 clusters, so in principle the minimum possible scan fraction is around 0.01.

Astronomy Data: This data set consists of 650K records of measurements on sky objects. For each object there are 29 measurements and we clustered the data into 10 clusters. Again, this was by no means an optimized clustering, and a larger number of clusters would imply a lower possible minimum fraction of data to scan. Results are shown in Table 3.

Here we introduce the notion of *Discounted Accuracy*. Our accuracy measure counts a full error if for K=50, we found 49 out of 50 true nearest neighbors. Discounted accuracy gives an error of $\frac{1}{50}$ in this case and thus provides more insight into what is happening. Note the discounted error results indicate that we

K	2	5	10	50
Accuracy	94.7%	90.0%	85.0%	78.6%
Discounted Accuracy	97.3%	96.4%	95.6%	95.2%
Fraction Scanned	16.8%	17.2%	17.4%	17.8%

Table 3: Accuracy and Fraction Scanned Results for 29-dimensional Astronomy Data

do indeed find most of the nearest neighbors. It is worthy to note that as K increases, we expect the model usefulness to decrease. In the trivial limit where K is large enough, all data must be scanned regardless of indexing scheme. Also note that the model was not optimized for the data, and we only used 10 clusters. We expect much better results as we increase number of clusters and build better optimized clustering models.

Investor Data Sets: The last data set we evaluated is derived from a financial database consisting on historical stock prices, market valuations and so forth. In all, we used 34 dimensions. The data consisted of 834K records. We tried two experiments: one with 20 clusters, and one with 47 clusters. Again these clusters were not optimized. The results for K=2 gave an error of 0.01 with 45% of the data scanned in case of 20 clusters. With 47 clusters the accuracy was 100% with 42% of the data scanned. As we increased K, performance deteriorated quickly. This, we believe is due to problems in fitting a model to data. We need to optimize the model. Also, the problems with numerical instability of the probability calculation and near-zero variances were prevalent in this data set. However, the results at least show than in principle the structure is useful (i.e. we did not have to scan 100% of the data).

7 Discussion

7.1 Determining the Number of Clusters

Throughout the discussion so far, we have not addressed the issue of determining the number of clusters required. From the perspective of indexing, the more clusters we have, the less data we are likely to have to scan. However, recall that determining which cluster to scan next requires a lookup into the model. If there are too many clusters, this lookup becomes too expensive. Consider the extreme case where each point in the database is its own cluster. In this case, no data will need to be scanned as the model identifies the result directly. However, the lookup into the model is now as expensive as scanning the entire database.

Generally, we use a small number of clusters (5 to 100). The cost of computing probabilities from the model in this case is fairly negligible. Note that with 5 clusters, assuming well-separated clusters, we can expect an 80% savings in scan cost. So not many clusters are needed. There are also clustering algorithms that choose K as part of the clustering session. In this case, we let the algorithm choose the most appropriate K to fit the data, so long as K does not get too large. The trade-off between model lookup time and data scan cost can be optimized on a per application basis.

7.2 Type of Queries Supported

The query records in nearest-neighbor applications can be either from the same distribution the data came from or from a different distribution. In either case, the DBIN algorithm is well defined since the probabilities discussed in Section 4 are for any fixed query points. Our current definition of stability in Section 5 assumes that the queries are generated from the same distribution. We have not analyzed the case where the query distribution is different. In the special case where the query items happen to be entries from the data table itself, then DBIN will find the result in the first cluster scan by definition. This is also generally true if the queries come from the same distribution that the data is drawn from and the distribution is stable. In these cases, we can analytically predict the expected gain from DBIN.

Assume that DBIN is using a clustering model with a set of clusters C, and let $|C_i|, i = 1, \ldots, K$, denote the size of C_i for $C_i \in C$. Let $|C| = \sum_{i=1}^{K} |C_i|$. If we define S_i to be the proportion of data that are members of C_i, i.e. $S_i = \frac{|C_i|}{|C|}$, then we expect that on an average query, only

$$\sum_{i=1}^{K} S_i^2$$

will be scanned. The reason for this is that with probability S_i the query will come from C_i, and scanning C_i is equivalent to scanning a portion S_i of the data.

Generally we expect that most queries in high dimensions will come from the same distribution as the data itself. Also, dimensions are not independent and some combinations of values may not be realistic. In a database of demographics, for example, one does not expect to see a "find similar" query on an entry for a record whose value for age is 3 and whose income is \$50K. One major benefit of DBIN is in the worst case where the data is not stable and a bad query is given, DBIN will predict that the nearest neighbor may reside in many clusters and the algorithm can switch to a sequential scan.

7.3 Other Indexing Methods

While in this paper we described a scheme that creates a clustered index based on cluster labels, we note that this is not strictly required. Viewed in general perspective, our scheme requires the ability to predict the probability of a point occurring in the intersection between a query ball and a region of the data space. Assuming we can compute the required probability, we can use our scheme in conjunction with any other scheme for partitioning the space into regions. Hence, we could use a statistical model of the data to derive a prioritization on which region to scan next and to rule out regions unlikely to contain data. The approach may be generalizable to other types of queries such as range queries. It is also possible to combine DBIN with indexing methods that employ bounding objects. For example, we could construct a minimum bounding rectangle or ellipsoid around each cluster found by DBIN. This might further improve performance.

7.4 Generalizations of DBIN

We conclude this section with a list of possible generalizations to our approach.

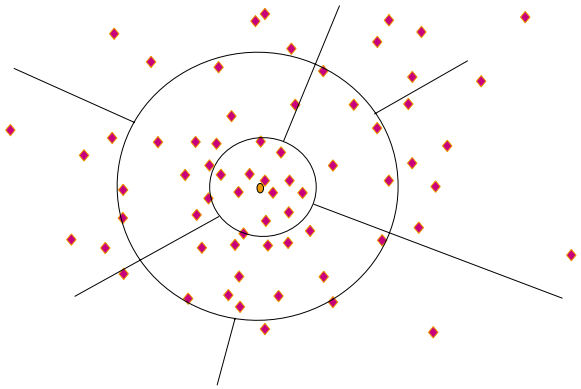

Figure 6: A Possible paging layout of data within a cluster

Optimizing Data Organization (paging) Within a Cluster: In this paper, we did not address the issue of how data should be ordered within a cluster. While the theory we developed essentially says it is possible to exploit the structure in data when data is clustered, within a cluster, it is a different question of what order should the data be scanned in. This becomes an issue in a practical DBMS as data within each cluster will have to be laid out on disk pages. It should be possible to use a stopping criterion within a cluster and avoid scanning all pages.

For example, consider a query point that happens to be near the center of a cluster. Clearly the data near the center should be scanned, while data within the cluster but further from the center may not have to. This suggests a disk organization that is similar to the one illustrated in Figure 6. The idea behind such a layout is that regions get bigger as they get farther from the center of the cluster. Furthermore, the central region should not be partitioned finely since theory tells us that data points close to the center are indistinguishable from each other by distance: i.e. they are all essentially equidistant from a query (see discussion in Section 5).

Distance Metrics: The scheme we developed for DBIN and the theory developed around it actually support a fairly wide family of measures. The only assumption that we make about the distance measure is that it is of the *quadratic normal form* [24], i.e. it can be written as:

$$\text{Dist}(x, q) = (x - q)^{\text{T}} D (x - q)$$

with D being a positive semi-definite matrix, i.e. for any vector x, $x^{\text{T}} Dx \geq 0$. Note that for Euclidean distance, D is diagonal with all entries being 1. Note that we can support weighted Euclidean distance:

$$\text{Dist}(X, Y) = \sqrt{\sum_{i=1}^{d} W_i (X_i - Y_i)^2}$$

as long as $W_i \geq 0$. So this admits nearest-neighbor queries based on a subset of features (feature selection).

Dealing with Updates to Data: We can accommodate additional data records easily since the clustering scheme used for scalable EM (SEM) is incremental [10]. First the new model needs to be updated, then the new data records need to be inserted into the old table while maintaining the sort by cluster membership. Record deletions can be accommodated by subtracting the records from the SEM model. Although possible, we have not tested performance of SEM when records are deleted. Record updates are handled by deletion of old followed by addition of new. Hence, as long as the clustering scheme being used in the preprocessing stage is incremental, DBIN should be able to handle change gracefully.

8 Conclusions

We presented DBIN, a scheme that exploits structure in data to derive a multi-dimensional index for approximate nearest-neighbor queries. DBIN efficiently finds probabilistic-approximate nearest-neighbors, i.e. nearest-neighbors with a given degree of confidence. We developed a probabilistic theory to support the method. We analyzed the nearest-neighbor query problem under the assumption that data is modeled by a mixture of Gaussians. We defined the notion of cluster stability which gives us the means to assess if a data set is amenable to our method. We derived two important results. The first is that nearest neighbor queries can be meaningful in high dimensions. The second is that it is possible to exploit the statistical structure of the data to construct a multi-dimensional indexing scheme. We derived a criterion for estimating the likelihood of payoff of scanning further, enabling a stopping criterion. The result is a new indexing algorithm based on density estimation. It provides a confidence level in the answer found so far. Because it is based on modeling the data content, DBIN provides sufficient information regarding the suitability of its indexing scheme to a given database. This can enable an optimizer to decide whether to invoke DBIN's indexing structure or opt for the sequential scan when appropriate.

We showed empirically that the proposed algorithm works when the data is stable. We used synthetic data to verify this and to study behavior when data is dramatically unstable. We also demonstrated for several real-world databases that our intuition holds that real data is not uniformly distributed or concentrated in a single cluster. Our method showed significant improvement over a sequential scan. DBIN has many properties that are desirable from a data mining perspective. It is an "anytime algorithm": it can provide the "best answer so far" whenever queried. The user can then stop the scan if the answer is satisfactory. It also provides a confidence level on the answer found so far. It can provide accurate estimates of how much work is left, allowing the user application to perform a cost-benefit analysis. It can utilize much of the existing SQL backend assuming clustered indices are supported. It leverages new work on scalable clustering techniques. Finally, we have found empirically that a full data scan is almost always avoided.

8.1 Future Work

The results indicate that visiting data in order of nearest clusters first is sound. In fact, we have found that in many of the instances when our stopping criterion does not cause us to stop after the first cluster, the

actual nearest neighbor turned out to be in the first scanned cluster after all. This suggests that it is possible to derive improved bounds for stopping earlier.

Other generalizations we are working on include generalization to discrete data. At its heart, the scheme we described does not require the data to be continuous. If a distance measure is defined over a discrete space (e.g. if two categorical fields have the same value, assign zero contribution and assign one if they are different), then as long as one clustered data using a similar criterion the same scheme should work. This assumes we have a probability model that assigns probability to events. For example, events in binary data happen to be corners on a hypercube defined by the space of all attributes.

Additional steps are needed to make DBIN into a practical system. One challenge associated with DBIN is that it must calculate probabilities that are very close to zero without underflowing. In these cases, we run into numerical stability issues. Numbers get small due to high dimensionality and to large numbers of records. While high-dimensionality is handled by the theory we develop, in practice we are left with computing vanishingly small probabilities. The probability algorithm used is this paper becomes numerically unreliable at about 50 dimensions. More robust schemes for computing these probabilities exist. We have just begun examining more robust methods for calculating these probabilities. This paper is intended mainly to validate the basic framework and theory.

We would like to generalize the theory to models other than Gaussians. Also, an evaluation of an implementation tied to a SQL backend that utilizes native indexing structures would be necessary to demonstrate practical feasibility.

Acknowledgements

We would like to thank Ilya Vinarsky for help on implementation and evaluation of empirical results presented here. We thank Paul Bradley and Cory Reina for help on deriving and using scalable clustering results on large databases.

References

[1] S. Berchtold, D. Keim B. Ertl, and H.-P. Kriegel. Fast nearest neighbor search in high-dimensional space. In *Proceedings 14th Int. Conf. on Data Engineering, Orlando, Fl*, 1998.

[2] S. Berchtold, C. Bohm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *ACM PODS, Tucson, AZ*, 1997.

[3] S. Berchtold, C. Bohm, and H.-P. Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. In *Proceedings of ACM SIGMOD, Seattle, WA*, pages 142–153, 1998.

[4] S. Berchtold and A. Keim. High-dimensional index structures: Database support for next decade's applications. Tutorial Notes: ACM SIGMOD-98 Conference on Management of Data, Seattle, 1998.

[5] S. Berchtold, D. Keim, and H-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd Conference on Very Large Databases, Bombay, India*, pages 28–39, 1996.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT), Jerusalem, Israel,1999*, 1998. To appear.

[7] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, 1995.

[8] P. Bradley and U. Fayyad. Refining initial points for k-means clustering. In *Proc. 15th International Conf on Machine Learning.* Morgan Kaufmann, 1998.

[9] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9 – 15, 1998.

[10] P. Bradley, U. Fayyad, and C. Reina. Scaling EM (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, Redmond, WA, USA, 1998.

[11] B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA, 1991.

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

[13] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[14] C. Faloutsos and K.-I Lin. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3(4):181–210, 1994.

[15] C. Faloutsos and K.-I Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceeding of ACM SIGMOD International Conference on Management of Data, San Jose*, pages 231–262, 1995.

[16] R. Farebrother. Algorithm as 204: The distribution of a positive linear combination of chi-square random variables. *Applied Statistics*, 32(3):332–337, 1983.

[17] U. Fayyad, P. Bradley, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining*, pages 194 – 198, 1998.

[18] E. Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *Biometrics*, 21(768), 1965.

[19] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions. Submitted for publication, 1998.

[20] A. Gutman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data, Atlantic City, New Jersey*, pages 322–331, 1997.

[21] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC'98. Proceedings of the Thirteenth annual ACM symposium on Theory of computing*, pages 604 – 613, 1998.

[22] N. Katayama and S. Satoh. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD International Conference on Management of Data, Atlantic City, New Jersey*, pages 322–331, 1997.

[23] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data, Tucson, Arizona*, 1997.

[24] A. Mathai and S. Provost. *Quadratic Forms in Random Variables.* Marcel Dekker, Inc, New York, 1992.

[25] J. T. Robinson. The K-D-B tree: A search structure for large multidimensional indexes. In *Proceedings of ACM SIGMOD International Conference on Management of Data, Ann Arbor, MI*, pages 10–18, 1981.

[26] D. W. Scott. *Density Estimation.* Wiley, New York, 1992.

[27] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc, of ACM SIGMOD, Seattle, WA*, pages 154–165, 1998.

[28] K. Shim, R. Srikant, and R. Agrawal. High-dimensional similarity joins. In *13th Int'l Conf. on Data Engineering*, 1997.

[29] B.W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, London, 1986.

[30] B. Thiesson, C. Meek, D. Chickering, and D. Heckerman. Computationally efficient methods for selecting among mixtures of graphical models. In *Proceedings of the Sixth Valencia International Meeting on Bayesian Statistics*, pages 223 – 240. Alcossebre, Spain, 1998.

[31] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. of the 12th Internatial Conference on Data Engineering, New Orleans*, pages 516–523, 1996.

[32] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2), 1997.

A Proof of Stability Theorems

This appendix contains the proofs of the Theorems 5.2, 5.3 and 5.4 in Section 5.

This is the proof of the stability result in the main text, Theorem 5.2.

Theorem A.1 (Pairwise Between Cluster Stability) *Assume q_d and $x_{d,i}$ are generated by cluster i and $x_{d,j}$ is generated by cluster j. If*

$$\frac{\|x_{d,i} - q_d\|^2}{E(\|x_{d,i} - q_d\|^2)} \longrightarrow_p 1 \tag{21}$$

$$\frac{\|x_{d,j} - q_d\|^2}{E(\|x_{d,i} - q_d\|^2)} \longrightarrow_p \delta > 1. \tag{22}$$

Then we say Clusters i and j are pairwise stable with parameter δ and for any $\epsilon > 0$

$$\lim_{d\to\infty} P(\|x_{d,j} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2) = 1 \tag{23}$$

Proof. Let $\mu_d = E(\|x_{d,i} - q_d\|^2)$. Let $V_d = \frac{\|x_{d,i} - q_d\|^2}{\mu_d}$ and let $W_d = \frac{\|x_{d,j} - q_d\|^2}{\mu_d}$. We know $V_d \to_p 1$ and $W_d \to_p \delta$. Thus

$$\frac{\|x_{d,j} - q_d\|^2}{\|x_{d,i} - q_d\|^2} = \frac{\mu_d\|x_{d,j} - q_d\|^2}{\mu_d\|x_{d,i} - q_d\|^2} = \frac{W_d}{V_d} \to_p \delta. \tag{24}$$

By definition of convergence in probability, for any $\epsilon > 0$

$$\lim_{d\to\infty} P\left[\left|\frac{\|x_{d,j} - q_d\|^2}{\|x_{d,i} - q_d\|^2} - \delta\right| \leq \epsilon\right] = 1 \tag{25}$$

So

$$\lim_{d\to\infty} P\left[\delta - \epsilon \leq \frac{\|x_{d,j} - q_d\|^2}{\|x_{d,i} - q_d\|^2)} \leq \delta + \epsilon\right] = 1. \tag{26}$$

which implies

$$\lim_{d\to\infty} (\|x_{d,j} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2) = 1 \tag{27}$$

□

To make nearest neighbor queries meaningful, we need the between cluster distance to dominate the within cluster distance for at least one cluster. This is Theorem 5.3 in the main text.

Theorem A.2 (Nearest neighbor well-defined) *Assume q_d and $x_{d,i}$ are generated by cluster i and $x_{d,j}$ is generated by cluster j. If cluster i is unstable with respect to itself and there exists a cluster j that is pairwise stable with i with parameter $\delta > 1$, then for any $\epsilon > 0$*

$$\lim_{d\to\infty} P[DMAX_d \geq (\delta - \epsilon)DMIN_d] = 1 \tag{28}$$

Proof. By conditions of the theorem

$$\frac{\|x_{d,i} - q_d\|^2}{E(\|x_{d,i} - q_d\|^2)} \longrightarrow_p 1. \tag{29}$$

and

$$\frac{\|x_{d,j} - q_d\|^2}{E(\|x_{d,i} - q_d\|^2)} \longrightarrow_p \delta. \tag{30}$$

By definition of minimum and maximum

$$\frac{DMIN_d}{E(\|x_{d,1} - q_d\|^2)} \leq \frac{\|x_{d,1} - q_d\|^2}{E(\|x_{d,1} - q_d\|^2)} \tag{31}$$

and

$$\frac{DMAX_d}{E(\|x_{d,1} - q_d\|^2} \geq \frac{\|x_{d,2} - q_d\|^2}{E(\|x_{d,1} - q_d\|^2)} \tag{32}$$

This implies

$$\frac{DMAX_d}{DMIN_d} \geq \frac{\|x_{d,2} - q_d\|^2}{\|x_{d,1} - q_d\|^2} \tag{33}$$

We know by the Theorem 5.2 that for any $\epsilon > 0$.

$$P[\frac{\|x_{d,2} - q_d\|^2}{\|x_{d,1} - q_d\|^2} \geq \delta - \epsilon] = 1. \tag{34}$$

From the last two statements,

$$P[\frac{DMAX_d}{DMIN_d} \geq \frac{\|x_{d,2} - q_d\|^2}{\|x_{d,1} - q_d\|^2} \geq \delta - \epsilon] = 1 \tag{35}$$

□

If for every cluster, the between cluster distance dominates the within cluster distance, then we know the nearest neighbor is within the same cluster. This is Theorem 5.4 in the main text.

Theorem A.3 (Nearest Neighbor in Cluster) *Assume q_d and $x_{d,i}$ are generated by cluster i. If cluster i is unstable with itself and and pairwise $\delta_{ij} > \delta > 1$ stable with every other cluster j, $j \neq i$, then q_d nearest neighbor was also generated by cluster i. Specifically for any point x_{d_j} from cluster $j \neq i$*

$$P[\|x_{d,j} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2] = 1. \tag{36}$$

Proof. By the assumptions of the theorem, for any sample j whose class is not i and $\epsilon > 0$

$$P[\|x_{d,j} - q_d\|^2 \geq (\delta_{i,j} - \epsilon)\|x_{d,i} - q_d\|^2 \geq (\delta - \epsilon)\|x_{d,i} - q_d\|^2] = 1 \tag{37}$$

B Stability Theorems Applied to Mixture of Gaussian

In this section we derive the stability conditions for a mixture of spherical Gaussians. Consider a mixture of spherical Gaussians in d dimensions where $m_{d,i}, \sigma_{d,i}^2 I, p_{d,i}$ are the means, covariance matrices, and mixture weights, i=1,...,K. The i^{th} Gaussian is called cluster C_i and there are K clusters.

If we let F_d be the sequence of distributions corresponding to a mixture of these Gaussians with increasing dimensionality. For each d, a sample $\{x_{d,i}\}_{i=1}^n$ of size N is generated from the mixture. Assume that for each d at least one sample point is generated by each Gaussian. Without loss of generality we will assume $x_{d,i}$, $i = 1, \ldots, K$ was generated by the ith Gaussian.

First let's assume that q_d is generated by a single Gaussian of the sample. Without loss of generality we can assume it is the first one. If we look at the distance between q and the sample points generated by the same Gaussian that generated q, they all become roughly the same distance apart. To show this note that the random vector $x_{d,1} - q_d$ has a Gaussian distribution with mean 0 and covariance matrix $2\sigma_d^2 I$. We know $\frac{||x_{d,1}-q_d||^2}{2\sigma_d^2}$ has a chi-square distribution with d degrees of freedom. By using this fact

$$E(\|x_{d,1} - q_d\|^2) = E(2\sigma_d^2 \frac{||x_{d,1} - q_d||^2}{2\sigma_d^2}) = 2\sigma_d^2 d. \tag{38}$$

and

$$var(\|x_{d,1} - q_d\|^2) = var(2\sigma_d^2 \frac{||x_{d,1} - q_d||^2}{2\sigma_d^2}) = 8\sigma_d^4 d. \tag{39}$$

Thus

$$\lim_{d\to\infty} \frac{var(\|x_{d,1} - q_d\|^2)}{E(\|x_{d,1} - q_d\|^2)^2} = \lim_{d\to\infty} \frac{8\sigma_d^4 d}{(2\sigma_d^2 d)^2} = 0 \tag{40}$$

and then by Corollary 5.1 we know q is unstable with respect to Cluster 1 .

Consider the distance between q from C_1 and the sample points generated by another Gaussian that did not generate q (say C_2). The random vector $x_{d,2} - q_d$ has a Gaussian distribution with mean $m_{d,2} - m_{d,1}$ and covariance matrix $2\sigma_d^2 I$. We know $\frac{||x_{d,2}-q_d||^2}{2\sigma_d^2}$ has a noncentral chi-square distribution with d degrees of freedom and noncentrality parameter $\frac{\|m_{d,2}-m_{d,1}\|^2}{2\sigma_d^2}$. The expected distance is then

$$\begin{aligned} E(\|x_{d,2} - q_d\|^2) &= E(2\sigma_d^2 \tfrac{||x_{d,2}-q_d||^2}{2\sigma_d^2}) \\ &= 2\sigma_d^2 (d + \tfrac{||m_{d,2}-m_{d,1}||^2)}{2\sigma_d^2}) \\ &= 2\sigma_d^2 d + ||m_{d,2} - m_{d,1}||^2 \end{aligned} \tag{41}$$

and the variance is

$$\begin{aligned} var(\|x_{d,1} - q_d\|^2) &= var(2\sigma_d^2 \tfrac{||x_{d,1}-q_d||^2}{2\sigma_d^2}) \\ &= 4\sigma_d^4 (2d + 4\tfrac{||m_{d,2}-m_{d,1}||^2}{2\sigma_d^2}) \\ &= 8\sigma_d^4 d + 8\sigma_d^2 ||m_{d,2} - m_{d,1}||^2. \end{aligned} \tag{42}$$

For a Gaussian cluster

$$\frac{var(\|x_{d,2} - q_d\|^2)}{E(\|x_{d,1} - q_d\|^2)^2} = \frac{8\sigma_d^4 d + 8\sigma_d^2 ||m_{d,2} - m_{d,1}||^2}{(2\sigma_d^2 d)^2} = \frac{2}{d} + \frac{2||m_{d,2} - m_{d,1}||^2}{\sigma_d^2 d} \tag{43}$$

This last quantity converges to 0 as d goes to infinity. So if the sequence of distributions for that cluster satisfies

$$\lim_{d\to\infty} \frac{||m_{d,2} - m_{d,1}||^2}{\sigma_d^2 d} = \delta. \tag{44}$$

Then by Corollary 5.2

$$\frac{\|x_{d,2} - q_d\|^2}{E(\|x_{d,1} - q_d\|^2)} \longrightarrow_p \delta. \tag{45}$$

Furthermore, if $\delta > 1$ than we know the between cluster distance dominates the within cluster distance. So by Theorem 5.2 we know the two clusters are stable. For example if we fix $\sigma_d = \hat{\sigma}$ and $m_{d,1}$ and $m_{d,2}$ such that $||m_{d,1} - m_{d,2}|| > \hat{\sigma}\sqrt{d}$ then the clusters will be stable.