CGL

Computational Geometric Learning

D2.1: Handling High-Dimensional Data

Frédéric Cazals Frédéric Chazal Christiane Lammersen

Ioannis Z. Emiris Bernd Gärtner Joachim Giesen

Charis Malamatos Günter Rote

CGL Technical Report No.: CGL-TR-01

Part of deliverable:	WP-2/D2.1
Site:	FUB
Month:	6

Project co-funded by the European Commission within FP7 (2010–2013)
under contract nr. IST-25582

1 Introduction and Overview

We give a survey and assessment of methods that are commonly used for dealing with high-dimensional data, mainly from the database community. We refer to the book by Samet [59] for a comprehensive overview.

In the following chapters, we will deal with the following topics.

- Dimension reduction
- Embedding methods
- Clustering
- Nearest neighbor methods

In order to store objects in a database, it is common to map every object to a *feature vector* in a (possibly high-dimensional) vector space. The feature vector then serves as the representation of the object. For example, to represent planar shapes within the unit square, say, one could subdivide the unit square into n pixels; the feature vector of a shape S would then be a 0-1-vector $\mathbf{x}(S)$ whose i-th component is 1 if and only if the i-th pixel is contained in the shape.

Performing database queries amounts to computing with feature vectors. In our example, finding the shape with smallest symmetric difference to a query shape Q can be done by computing the nearest neighbor $\mathbf{x}(S)$ to $\mathbf{x}(Q)$ w.r.t. the Hamming distance. For 0-1-vectors as in this case, minimizing the Hamming distance is equivalent to minimizing the Euclidean distance.

In high-dimensional spaces, many geometric data structures fail to work well. This *curse of dimensionality* can be approached in several ways. Section 2 describes methods of *Dimension Reduction*: mapping the high-dimensional feature space to a lower-dimensional space while preserving (most of) the relevant properties. Dimension reduction is often used as a preprocessing step in nearest neighbor search, see Section 5 *Embedding Methods*, which are described in Section 3, are complementary to the methods of Section 2: In contrast to the dimension reduction techniques of Section 2, which use linear mappings (projections), here one focuses also on non-linear mappings.

Section 4 discusses *Clustering Methods*. Clustering is an important data analysis and data preprocessing tool, which is widely used and studied in the database community. Since it is based mostly on pairwise distances, the methods are inherently independent on the dimension of the data, and thus equally suited to high- and low-dimensional data. Clustering is also used as a tool in nearest-neighbor data structures (Section 5.1).

Section 5 describes the data-structure view to high-dimensional *Nearest Neighbor Searching* (NNS). We consider point sets and point queries, which is the standard version of the problem in Computational Geometry. Many instances of NNS reduce to this case. We put the emphasis on *approximate* NNS, in order to be able to handle high-dimensional data. The main question is how to preprocess the input point set into an external-memory data structure, or *index*, so that queries can be efficiently reported. Section 5 focuses on three state-of-the-art paradigms to construct indexes in high dimensions, which offer provably correct results. However, their strong performance is typically corroborated only by experimental evidence, since complexity analyses are very difficult. This evidence shows that average-case behavior is strong, although there may exist some rare bad cases. We juxtapose these approaches to a few representative data-structures and implementations from the Computational Geometry community.

We conclude with a brief summary in Section 6.

2 Dimension Reduction

As mentioned in the introduction, objects in a database are often represented as feature vectors.

If feature vectors are high-dimensional, many data structures for similarity queries and other tasks based on spatial properties of the feature vectors fail to work well. This *curse of dimensionality* is in part due to (somewhat counterintuitive) properties of high-dimensional space. For example, most of the volume of a high-dimensional ball or a box is very close to the boundary, meaning that hierarchical volume subdivision methods typically need to make many "inefficient" subdivisions close to the boundary. Already in two dimensions, representing a polygon up to a fixed error by a quadtree requires a number of nodes proportional to the circumference of the polygon [37].

Dimension reduction computes a mapping f from the high-dimensional feature space $\mathbf{R}^n$ to a lower-dimensional space $\mathbf{R}^k$, with the goal of preserving the properties of the feature vectors that are relevant for the application at hand. A spatial data structure is then built on top of the transformed vectors in $\mathbf{R}^k$, and query results are refined, if necessary, using information from the original feature vectors. In this section, we focus on the main methods (and some of their variations) that have been used for dimension reduction in the database community.

Contraction, or the Pruning Property. A desirable property of the transformation function $f : \mathbf{R}^n \to \mathbf{R}^k$ is that it is *contractive*, i.e. (in the Euclidean metric) that

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq \|\mathbf{x} - \mathbf{y}\|.$$

This is also called the *pruning property*. The significance of this property is best seen in the context of nearest neighbor search. Suppose that we want to find for a query vector $\mathbf{q} \in \mathbf{R}^n$ all vectors $\mathbf{x}$ in the database such that $\|\mathbf{x}-\mathbf{q}\| \leq r$. If we perform the same query in the transformed space, i.e. find all $\mathbf{x}$ such that $\|f(\mathbf{x})-f(\mathbf{q})\| \leq r$, we can be sure under contraction that this set also contains all $\mathbf{x}$ for which $\|\mathbf{x}-\mathbf{q}\| \leq r$, i.e. we have no *false dismissals*.

It is important to understand that f may be contractive w.r.t. the Euclidean metric, but not w.r.t. other metrics. In this section, we focus on the Euclidean metric as a measure of similarity.

Dimension reduction can be done in a *data-dependent* or a *data-independent* way. In the former case, the mapping f depends on the data, with the clear advantage that f can be tailored to the data, and the clear disadvantage that f typically needs to be recomputed if data changes (even slightly). In the latter case, f only depends on the parameters n and k, and possibly a random resource.

Simple dimension reduction techniques based on projections to fixed coordinates are self-evident and can be useful (in particular, they are contractive), but their power is obviously limited. Most notably, one can try to find the "most relevant" of the n coordinates, and then ignore all other coordinates, resulting in a set of transformed vectors in dimension 1. This approach generalizes to keeping the k most relevant coordinates. In the following, we focus on the more advanced standard methods.

2.1 Data-dependent methods

The main idea of data-dependent methods is to discover lower-dimensional structure(s) in the high-dimensional data. Recent (and nonlinear) such techniques are in more detail discussed in Section 3. Here, we focus on techniques to discover *linear* structures of lower dimension.

Principal component analysis. PCA is a classic technique [55] for computing the k-dimensional flat F that best approximates the n-dimensional set of feature vectors; best in the sense that the sum of squared distances of each point to its projection onto F is minimized. The problem can be solved using *singular value decomposition*, a standard technique from linear algebra [30]. The transformed vectors are precisely the projections of the feature vectors onto the computed flat F (expressed in a basis of this k-dimensional subspace). Singular value decomposition is also the basis of *latent semantic indexing*, a dimension reduction technique used for indexing large collections of documents [21].

Dynamic SVD. PCA is best suited for static data. As soon as feature vectors change, the coordinate transformation that "rotates" F into $\mathbf{R}^k$ needs to be recomputed which is expensive. An attractive idea pursued in [56] is the following: (i) Under feature vector changes, the current flat F is updated only when the query performance (measured by some heuristic criterion) has dropped too much. (ii) To perform the update, the SVD is not done with the full set of feature vectors, but with only a few *aggregate values* instead. The aggregation is done using the search structure built over the feature vectors, based on the transformation to $\mathbf{R}^k$. For example, if the vectors are stored in the leaves of a tree, aggregation can be done by fixing a level in the tree, and then replacing all vectors in each subtree hanging off that level with the centroid of these vectors. Experiments show that the resulting SVD is not much worse than the full SVD in suitable applications [56].

RCSVD. Another problem with PCA is that it generates only *one* flat F. In many situations, the data exhibits linear substructures of lower dimension, but this structure may consist of several flats. In such a situation, we are confronted with a clustering problem in the first place, see Section 4. The approach of the RCSVD algorithm [64] is to (i) perform a PCA of the full set of feature vectors; (ii) cluster the transformed data into non-overlapping parts; (iii) recursively handle the parts, until no further dimension reduction can be achieved.

2.2 Data-independent methods

Data-independent methods do not rely on characteristics of a specific set of data, but rather on characteristics of the data source. For example, we may know that we are dealing with time series data such as daily stock prices over the last year.

Discrete Fourier transform. The Discrete Fourier transform (DFT) is a unitary transformation of the complex space $\mathbf{C}^n$. It expresses a sequence of n complex numbers (in this context called a "signal") as a linear combination of n-th roots of unity. The roots of unity are "phases", and the coefficients in the linear combination are "amplitudes". The idea is to decompose the signal into periodic signals of varying phases and amplitudes. The DFT can be computed using the *Fast Fourier Transform* (FFT) in $O(n \log n)$ time rather than the obvious $\Theta(n^2)$ for matrix-vector multiplication.

In a seminal paper, Agrawal, Faloutsos and Swami have introduced the DFT as a tool for dimension reduction in the context of time series data [3]. Their observation was that in many cases, only a few leading phases (the ones of lowest frequency) are necessary in order to accurately represent time series data. Thus, dimension reduction is very simple here: (i) perform the DFT; (ii) in the transformed space, use the simple dimension reduction scheme of projecting onto the leading coordinates.

Random projections. We have already made the point that projections to a fixed set of k coordinates are usually not very effective in keeping (metric) properties of the feature vectors. In geometric terms, these are projections to axis-aligned k-flats. But maybe surprisingly, projecting to *random* k-flats is much better. The classic *Johnson-Lindenstrauss-Lemma* (in the version of Matoušek [48]) states the following. If there are N feature vectors, then the projection f to a random k-dimensional subspace, for $k = O(\log(N)/\varepsilon^2)$, satisfies

$$(1-\varepsilon)\|\mathbf{x}-\mathbf{y}\|^2 \leq \|f(\mathbf{x})-f(\mathbf{y})\|^2 \leq (1+\varepsilon)\|\mathbf{x}-\mathbf{y}\|^2,$$

with high probability. Thus, f is not contractive, but approximately contractive, and even approximately distance-preserving. The reduction in dimension is *exponential*. Achlioptas has made this result useful for database applications, by showing that the projection to a random subspace can be replaced with a linear transformation that is realized by a $k \times N$ matrix with elements chosen randomly from the set $\{-\sqrt{3}, 0, \sqrt{3}\}$ [1].

The constants behind the big-O in the Johnson-Lindenstrauss-Lemma are not small [48]. Despite this, it was shown that random projections behave well in practical applications, even for small values of k [10].

3 Embedding Methods

Classically, embedding methods are meant to ease the calculation of (costly) distances between objects. They typically embed the objects into a vector space within which a distance metric approximating the original one can be used [59]. On a pairwise basis, the quality of an embedding is measured by the distortion incurred by the distance metric. In the context of nearest neighbor searches (see Section 5), one can assess the symmetric difference between the exact k-nearest neighbors, and those reported in the embedded space.

As opposed to this body of work, our focus in this survey is on embedding methods for learning theory. That is, present embedding methods in the context of non-linear dimensionality reduction, the ultimate goal being to learn a topological space from a sampling, and to report features of that space.

3.1 Embedding via spectral methods

Many of the most popular approaches to embed data in Euclidean spaces boil down to *spectral methods*. Note that the term *spectral method* is ambiguous and used differently within different communities. Here we use it in the sense of data analysis similar as van der Maaten et al. did [65]. That is, for us in a spectral method, a symmetric matrix is derived from the data and the solution to a given optimization problem can be obtained from the eigenvectors of this matrix. The geometric optimization problems that lead to a spectral technique are mostly of a *least squares* nature and include the following classical (and archetypical) problems:

(1) Find the k-dimensional subspace that approximates the point cloud best in a least squares sense.

(2) Find the embedding of the point cloud in k-dimensional space that preserves the distances between the points best possible in a least squares sense.

The first problem is the well-known *principal component analysis* (PCA, see 2.1) as it asks for the principal directions (components) of the data. It essentially is a data quantization technique: every data point gets replaces by its projection onto

the best approximating k-dimensional subspace. The loss incurred by the quantization is the variance of the data in the directions orthogonal the best approximating k-dimensional subspace. As long as this variance is small PCA can also be seen as *denoising* the original data. Many machine learning techniques including clustering, classification and semi-supervised learning [36], but also near neighbor indexing and search can benefit from such a denoising.

The second problem is called *multi-dimensional scaling (MDS)*. An important application of MDS is visualization of the point cloud data: the data points get embedded into two- or three-dimensional space, where they can be directly visualized. The main purpose of visualization is to use the human visual system to get insights into the structure of the point cloud data, e.g., the existence of clusters or—for data points labeled with discrete attributes—relations between this attributes. MDS visualization remains to be a popular tool for point cloud data analysis, but of course a lot of information will get lost (and in general cannot be restored by the human visual system) if the *intrinsic dimension* of the data points is larger than three.

Recently the focus in point cloud data analysis shifted: more emphasis is put on detecting non-linear features in the data, although processing the data for visual inspection still is important. What drives this shift in focus is the insight that most features are based on *local* correlations of the data points, but PCA and MDS both have only a global view on the point cloud data. The shift towards local correlations was pioneered by two techniques called *ISOMAP* [63, 20] and *Locally Linear Embedding (LLE)* [57, 58]. It is important to note that focusing on local correlations does not mean that one loses the global picture: for example the global intrinsic dimension of the data can be estimated from local information, whereas it is often (when the data are embedded non-linearly) not possible to derive this information from a purely global analysis. ISOMAP and LLE and their successors (some of which we will also discuss here) can be used both for the traditional purposes data quantization and data visualization. In general they preserve more information of the data (than PCA and MDS) while achieving a similar quantization error or targeting the same embedding dimension for data visualization, respectively.

Advantages of spectral methods. In this section, we have focused on a set of well-known methods, which follow a common thread as they ultimately resort to spectral analysis. Consider a point cloud P sampled from a manifold M embedded in $\mathbb{R}^d$. All methods that we consider intend to find the best embedding of the dataset P into a Euclidean space $\mathbb{R}^k$ with respect to some quadratic constraint reflecting different geometric properties of the underlying manifold M. The embedding of the data that minimizes the quadratic constraint can then be interpreted as the best k-dimensional embedding of the data with respect to the geometric property we aim to preserve. In most cases, the quadratic minimization problem boils down to a general eigenvalue problem ensuring to find a global minimum. Moreover, the embedding can be found by easy-to-implement polynomial-time algorithms.

This provides a substantial advantage over iterative or greedy methods based upon Expectation-Maximization like algorithms that do not provide guarantees of global optimality. In particular, for quite large data sets, the methods we consider still provide results when iterative and greedy methods fail due to complexity issues. Another advantage of spectral methods is that the quadratic objective function leads to a measurement of the quality of the embedding. At last, spectral methods have been widely used and studied in many applications areas (graph theory, mesh processing [70]) giving rise to a large amount of efficient theoretical and algorithmic tools that can be used for dimensionality reduction and data embedding.

3.2 Non-linear embedding methods

In this section, we briefly present a set of quite famous embedding methods that have interesting geometric interpretations. They also have the advantage of leading to easy to implement polynomial-time algorithms that prove more efficient with larger data sets than the ones usually involved in iterative or greedy methods (like e.g. the ones involving EM or EM-like algorithms). We also discuss the guarantees provided by these methods. Although this is not always the case with databases, in the following the considered data sets P are assumed to be in an Euclidean space $\mathbb{R}^d$ and sampled on/around a possibly unknown smooth manifold M of dimension $k < d$. However most of the methods mentioned below just rely on a notion of distance between neighboring data points. The common thread of these few methods is that they all aim to find an embedding $\hat{P} \subset \mathbb{R}^k$ of the data set P minimizing a quadratic functional $\phi(\hat{P})$ that intends to preserve (local) neighborhood information between the sample points.

Maximum Variance Unfolding (MVU). Classical methods like PCA (Principal Component Analysis) and MDS (MultiDimensional Scaling) perform poorly when data points are not close to an affine subspace because they are both based on an inherent linearity assumption. Especially, both methods fail when the data points are close to a "curled up" linear space—the most famous example is the so called Swiss roll data set, points sampled densely from a curled up planar rectangle in $\mathbb{R}^3$. The idea behind *maximum variance unfolding (MVU)*, introduced by Weinberger and Saul [66, 68, 67], is to unfold the data, i.e., to transform the data set to a locally isometric data set, that is closer to an affine subspace. The unfolding aims at maximizing the distance between non-neighboring points (after some choice of neighborhood) while preserving the distances between neighboring points.

Technically MVU proceeds as follows: let $D = (d_{ij} = \|p_i - p_j\|^2)$ be the symmetric $(n \times n)$-matrix of pairwise distances. Choose a suited neighborhood for each point in P (Weinberger and Saul choose the symmetric κ-nearest neighbors) and let the indicator variable n_{ij} be 1 if either p_i is in the neighborhood of p_j or p_j is in the neighborhood of p_i, and 0 otherwise. From D an *unfolding*, a positive semi-definite $(n \times n)$-matrix $K = (k_{ij})$ (interpreted as the Gram matrix of the unfolded point set) is computed through the following semi-definite program (SDP)

Maximize the trace of K subject to

(1) K is positive semi-definite

(2)
$$\sum_{i,j=1}^{n} k_{ij} = 0$$

(3)
$$k_{ii} - 2k_{ij} + k_{jj} = d_{ij} \text{ for all } (i,j) \text{ with } n_{ij} = 1$$

From K a lower dimensional embedding can be computed as described for MDS.

Locally Linear Embedding (LLE). LLE is a method introduced in [57, 58] that intends to take into account the local linearity of the underlying manifold M to perform the reduction of dimension. In a first step, LLE discards pairwise distances between widely separated points by building a neighborhood graph G. The goal of this first step is to connect only close points of P so that the neighbors of each vertex p_i in G are contained in a small neighborhood of p_i which is close to the tangent space of the underlying manifold M at p_i. To take this local linearity

into account, LLE computes for each vertex p_i of the graph its best approximation as a linear combination of its neighbors. More precisely, one computes a sparse matrix of weights $W_{i,j}$ that minimize the quadratic error

$$\varepsilon(W) = \sum_{i=1}^{n} \|p_i - \sum_{j \in N(p_i)} W_{i,j} p_j\|^2$$

where $N(p_i)$ is the set of the vertices that are connected to p_i in G. This is a simple least square problem. Solving it with the additional constraint

$$\forall i, \quad \sum_{j \in Ngb(p_i)} W_{i,j} = 1$$

makes the weights invariant to rescaling, rotations and translations of the data (the weights thus characterize intrinsic properties of the data). The weights matrix is then used to perform the dimensionality reduction: given $k < d$, the points p_i are mapped to the points $\hat{p}_i \in \mathbb{R}^k$ that minimize the quadratic function

$$\Phi(\hat{p}_i) = \sum_i \|\hat{p}_i - \sum_j W_{i,j} \hat{p}_j\|^2$$

This quadratic minimization problem classically reduces to solving a sparse $n \times n$ eigenvalue problem. To provide satisfactory result, the data have to be sufficiently dense to ensure that the neighbors of a given point provide a good approximation of the tangent space of M. Moreover, even if the data are dense enough, the choice of the neighbors may also be awkward: choosing a too small or too large neighborhood may lead to very bad estimates of the tangent space.

ISOMAP. ISOMAP is a version of MDS introduced in [63, 20], where the matrix of Euclidean distances is replaced by the matrix of the geodesic distances between data points on M. In a first step, ISOMAP builds a neighborhood graph such that the distances between points of P in the graph are close to the geodesic distances on M. Once the geodesic distance matrix has been built, ISOMAP proceeds like classical MDS to embed P in $\mathbb{R}^k$.
One of the advantage of ISOMAP is that it provides convergence guarantees. First, it can be proven that if the data are sufficiently densely sampled on M, the distance on the neighbor graph is close to the one on M [19, 49, 28]. Nevertheless, in practice robust estimation of geodesic distances on a manifold is an awkward problem that requires rather restrictive assumptions on the sampling. Second, since the MDS step in the ISOMAP algorithm intends to preserve the geodesic distances between points, it provides a correct embedding if M is isometric to a convex open set of $\mathbb{R}^k$. It appears that ISOMAP is not well-suited to deal with data on manifolds M that do not fulfill this hypothesis. Nevertheless some variants (conformal ISOMAP [20]) have been proposed to overcome this issue. Note also that ISOMAP is a non local method since all geodesic distances between pairs of points are taken into account. As a consequence ISOMAP involves a non-sparse eigenvalue problem which is a main drawback of this method. To partly overcome this difficulty some variant of the algorithm using landmarks have been proposed in [20].

Laplacian Eigenmaps. This method introduced in [8, 9] follows the following general scheme: first a weighted graph G with weights $W_{i,j}$ is built from the data. Here the weights measure closeness between the points: intuitively the bigger $W_{i,j}$ is, the closer p_i and p_j are. A classical choice for the weights is given by the Gaussian kernel $W_{i,j} = \exp(-\frac{\|p_i - p_j\|^2}{4\sigma})$, where σ is a user-defined parameter[1]. Second the

[1] To obtain a sparse matrix W the values of $W_{i,j}$ that are smaller than some fixed small threshold are usually set to 0.

graph G is embedded into $\mathbb{R}^k$ in such a way that the close connected points stay as close as possible. More precisely the points p_i are mapped to the points $\hat{p}_i \in \mathbb{R}^k$ that minimize

$$\phi(\hat{P}) = \sum_{i,j} \|\hat{p}_i - \hat{p}_j\|^2 W_{i,j}.$$

There is an interesting and fundamental analogy between this discrete minimization problem on the graph G and a continuous minimization problem on M. Indeed, it can be seen that minimizing ϕ on the functions defined on the vertices of G corresponds (in a discretized version) to minimizing $\int_M \|\nabla f\|^2$ on the space of functions f defined on M with L^2 norm $\|f\|_{L^2}^2 = \int_M \|f\|^2 = 1$. From the Stokes formula, this integral is equal to $\int_M \mathcal{L}(f)f$, where $\mathcal{L}$ is the Laplace-Beltrami operator on M and its minimum is realized for eigenfunctions of $\mathcal{L}$. Similarly the minimization problem on G boils down to a general eigenvector problem involving the Laplacian of the graph. Indeed the Laplace operator on G is the matrix $L = D - W$, where D is the diagonal matrix $D_{i,i} = \sum_j W_{i,j}$. It can be seen as an operator acting on the functions f defined on the vertices of G by subtracting from $f(p_i)$ the weighted mean value of f on the neighbors of p_i. By a classical computation, one can see that $\phi(\hat{P}) = tr(\hat{P}^T L \hat{P})$, where $\hat{P}$ is the $n \times k$ matrix with i-th row given by the coordinates of $\hat{p}_i$. It follows that, given $k > 0$, the minimum of ϕ is deduced from the computation of the $k+1$ smallest eigenvalues of the equation $Ly = \lambda Dy$ (the smallest one corresponding to the eigenvalue 0 has to be removed). The analogy between the discrete and continuous setting extends to the choice of the weights of G: choosing $W_{i,j} = \exp(-\frac{\|p_i - p_j\|^2}{4\sigma})$, where σ is a user-defined parameter, allows to interpret the weights as a discretization of the heat kernel on M [8]. From the side of the guarantees, the Laplacian eigenmaps only involve intrinsic properties of G so they are robust to isometric perturbations of the data. Moreover, the relationship with the Laplacian operator on M provides a framework leading to convergence results of L to the Laplace operator on M [9].

Hessian Eigenmaps (HLLE). ISOMAP provides guarantees when the unknown manifold M is isometric to a convex open subset of $\mathbb{R}^k$. Although the hypothesis of being isometric to an open subset of $\mathbb{R}^k$ seems to be rather reasonable in several practical applications, the convexity hypothesis appears to be often too restrictive. HLLE is a method introduced in [22] intending to overcome this convexity constraint. The motivation of HLLE comes from a rather elementary result stating that if M is isometric to a connected open subset of $\mathbb{R}^k$ then the null-space of the operator defined on the space of $\mathcal{C}^2$-functions on M by

$$\mathcal{H}\colon f \to \int_M \|\operatorname{Hess} f(m)\|^2 dm$$

where $\operatorname{Hess} f$ is the Hessian of f, is a $(k+1)$-dimensional space spanned by the constant functions and the "isometric coordinates" of M. It is thus appealing to estimate this null space in order to recover these isometric coordinates to map M isometrically on an open subset of $\mathbb{R}^k$. To do this the algorithm follows the same scheme as LLE and the estimation of the null-space of $\mathcal{H}$ reduces to an eigenvalue computation of a sparse $n \times n$ matrix. As a consequence HLLE allows to process embedding for a larger class of manifolds M than ISOMAP. The quality of the reduction is obviously closely related to the quality of the approximation of the kernel of the operator $\mathcal{H}$. Nevertheless, it is important to notice that the algorithm involves the estimation of second order differential quantities for the computation of the Hessian while LLE requires only first order ones to approximate the tangent space of M. To be done efficiently this usually needs a very dense sampling of M.

At last, note that HLLE is the same as Laplacian Eigenmaps where the Laplacian operator has been replaced by $\mathcal{H}$.

Diffusion Maps. Diffusion maps [14] provide a method for embedding based upon Markov random walks on a weighted graph G reflecting the local geometry of P . The graph G is built in a similar way as for Laplacian Eigenmaps: the larger is the weight of an edge, the "closer" are its endpoints. In particular G can be built using the discretization of the heat kernel on M (see section 3.2). From the weights matrix W one constructs a Markov transition matrix Π by normalizing the rows of W

$$\Pi_{i,j} = \frac{W_{i,j}}{d(p_i)} \quad \text{where} \quad d(p_i) = \sum_k W_{i,k} \quad \text{is the degree of the vertex} \quad p_i$$

$\Pi_{i,j}$ can be interpreted as the probability of transition from p_i to p_j in one time step. The term $\Pi_t(i,j)$ of the successive powers Π^t of Π represent the probability $\Pi_t(p_i,p_j)$ of going from p_i to p_j in t steps. The matrix Π can be seen as an operator acting on the probability distributions supported on the vertices of G. It admits an invariant distribution ϕ_0 defined by $\phi_0(p_i) = \frac{d(p_i)}{\sum_j d(p_j)}$. The idea of diffusion maps is thus to define a metric between the points of P which is such that at a given $t > 0$ two points p_i and p_j are close if the conditional distributions of probability $\Pi_t(p_i,.)$ and $\Pi_t(p_j,.)$ are close. The choice of a weighted L^2 metric between the conditional distributions allows to define a *diffusion metric* between the points of P

$$D_t^2(p_i,p_j) = \sum_k \frac{(\Pi_t(p_i,p_k) - \Pi_t(p_j,p_k))^2}{\phi_0(p_k)}$$

which is closely related to the spectral properties of the random walk on G given by Π. Intuitively, two points p_i and p_j are close if there are many paths connecting them in G. Note that the parameter t representing the "duration" of the diffusion process may be interpreted as a scale parameter in the analysis. Given k and $t > 0$, the *diffusion map* provides a parameterization and an embedding of the data set which minimizes the distortion between the Euclidean distance in $\mathbb{R}^k$ and the diffusion distance D_t. The diffusion map is obtained from the eigenvectors of the transition matrix Π and the eigenvalues to the power t of the transition matrix. The diffusion maps framework reveals deep connections with other areas (such as spectral clustering, spectral analysis on manifolds,...) that open many questions and make it an active research area. For a more detailed presentation of diffusion maps and its further developments the reader is referred to [14, 15, 44].

4 Clustering

During the past decades, clustering algorithms have been extensively studied in the database community. It goes beyond the scope of this report to give a comprehensive overview of the vast available literature. For this reason, we will focus our summary only on the milestones in this research area.

One of the most widely used clustering algorithms is Lloyd's k-means algorithm [46, 26, 47]. This algorithm is based on two observations: (i) Given a fixed set of centers, we obtain the best clustering by assigning each point to the nearest center and (ii) given a cluster, the best center of the cluster is the center of gravity (mean) of its points. Lloyd's algorithm repeatedly applies these two local optimizations steps to the current solution until no more improvement is possible. It is known that the algorithm converges to a local optimum [61], and the quality of the computed solution is sensitive to the choice of the starting centers.

A popular density-based algorithm is DBSCAN (Density Based Spatial Clustering of Applications with Noise) [23]. This is a heuristic that works based on the following two assumptions: (i) Inside of a cluster the density of points is higher than the density outside of the cluster and (ii) within areas of noise the density of points is lower than the density inside of any cluster. The key idea of the algorithm is now that each cluster contains at least one so-called *core point* and a cluster consists of all the points that are *density-reachable* from such a core point. Here, a point is called core point if the ball with certain radius centered at this particular point contains more than a certain threshold of points, i.e., the density in some local neighborhood of the point exceeds some threshold. Furthermore, a point p is called directly density-reachable from a point q if q is a core point and p is contained in the local neighborhood of q. By canonically extending this definition, a point p is called density-reachable from a point q if there is a chain of directly density-reachable points that starts in q and ends in p. Finally, points that are neither density-reachable from core points nor core points themselves are considered as noise. The advantages of DBSCAN are that it is able to discover clusters of arbitrary shape and that it does not require the number of clusters as input parameter.

A well-known medoid-based algorithm is CLARANS (Clustering Large Applications based on RANdomized Search) [52]. A medoid is similarly defined as a mean of a cluster but it is always a member of the input points. More precisely, a medoid is a point of a cluster whose average distance to all the points in the cluster is minimal. The heuristic CLARANS is based on the two medoid-based methods PAM (Partitioning Around Medoids) and CLARA (Clustering LARge Applications) developed by Kaufman and Rousueeuw [40]. PAM is a local search heuristic that starts with k arbitrary medoids and iteratively swaps one medoid and one non-medoid such that changing the status of these points decreases the total sum of the distances of the points to their closest medoids the most. This local improvement step is repeated until the algorithm reaches a local minimum of the clustering cost. In order to reduce the running time, Kaufman and Rousueeuw developed a randomized version of PAM, which they called CLARA. Basically, CLARA draws multiple random samples of the input points, runs algorithm PAM on each sample set, and outputs the clustering with the lowest clustering cost (in terms of the sum of distances of the points to the closest medoid). Like CLARA the algorithm CLARANS is also a randomized version of PAM. In each local improvement step, CLARANS does not consider all possible swaps of one medoid and one non-medoid like PAM does, but it only considers a randomly chosen subset of all possible swaps. CLARANS computes a certain fixed number of local minima in this manner and outputs the clustering with the minimum clustering cost among all these local minima.

One of the earliest and best known practical clustering algorithms for data streams is BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [71]. BIRCH is a heuristic which exploits the observation that the point space is usually not uniformly occupied. It scans the given set of input points once and computes a pre-clustering by summarizing dense regions of points by their so-called clustering features. Such a clustering feature consists of the number of points in the region, the center of gravity, and the sum of squared distances to the origin. Thereby, the problem of clustering the original input point set is reduced to the problem of clustering the set of summaries, which is much smaller than the original point set. The pre-clustering is then clustered by using an agglomerative (bottom-up) clustering algorithm. In this process, the algorithm uses the clustering features to calculate the intra-cluster distances. BIRCH successively merges the closest pair of clusters until the desired number of clusters is obtained.

Another well-known heuristic which uses a hierarchical clustering approach is

CURE (Clustering Using REpresentatives) [34]. This algorithm is based on the following two observations: (*i*) The shape of a cluster might not be spherical and (*ii*) outliers are typically further away from the mean of a cluster. Like BIRCH, CURE uses an agglomerative clustering approach. It starts with each point as one single cluster and then successively merges the two clusters with the closest pair of representative points until the desired number of clusters is obtained. In contrast to BIRCH, CURE is not centroid-based but represents each cluster by a certain constant number of well scattered points. By representing each cluster by multiple points, the detection of clusters with non-spherical shape is improved. In order to dampen the effect of outliers, each representative point is shrunken towards the mean of its cluster by some certain fraction. Furthermore, to speed-up computations, CURE applies random sampling and a partitioning approach.

Another well-known clustering algorithm for data streams is the streaming implementation of algorithm LSearch from O'Callaghan et al. [54] and Guha et al. [31]. This algorithm partitions the input stream into chunks and computes for each chunk a k-means clustering solution using a local search algorithm from Guha et al. [32]. Finally, the local search algorithm is applied once more on the union of the solutions for the chunks to obtain a k-means clustering for the whole input stream. The local search algorithm of Guha et al. [32] takes advantage of the relationship between the k-means clustering problem and the uniform facility location problem. In the latter problem, the number of facilities (or cluster centers) is not limited, but some cost has to be paid for each used facility. Now, the algorithm of Guha et al. is based on the observation that if the opening cost of a facility increases, then the number of facilities of an optimal solution tends to decrease. Hence, to solve the k-means problem, the algorithm of Guha et al. performs a binary search on the opening cost of a facility to find a cost that gives the desired number of cluster centers. During the binary search, each facility location problem is solved by starting with an initial solution that is obtained by a simple non-uniform sampling approach and then refining this solution by making local improvements.

A relatively new research area which is of great interest for our project is clustering of uncertain data. This is motivated by the fact that most of the real world datasets contain uncertain, imprecise, or incomplete data. Typical examples are measurements of sensor networks or datasets arising from record linkage across multiple data sources.

In recent publications, some traditional clustering approaches have been extended so that they can handle uncertain data. For instance, Chau et al. [12] and Ngai et al. [53] extended Lloyd's k-means algorithm, and Kriegel and Pfeifle [42, 43] and Xu and Li [69] extended the density-based clustering algorithm DBSCAN for handling uncertainty.

Very recently, Günnemann et al. [35] developed a subspace clustering heuristic for uncertain data. Subspace clustering was introduced for high-dimensional input data whose clusters appear in low-dimensional subspaces rather than in the full-dimensional ambient space. The input of the algorithm of Günnemann et al. is a set of probabilistic points from a Euclidean space where each point is formalized as a probability distribution function. The output is a set of subspace clusters. Each such cluster is defined as a set of relevant dimensions and a subset of the input points where each point is associated with a membership degree. This means, a point can belong to several clusters and a member ship degree indicates how relevant a particular point for a particular cluster is. The key idea of the algorithm is to approximate subspace clusters via hypercubes. In the relevant dimensions of the subspace cluster, the extent of the hypercube is limited by a fixed maximal width, while, in the non-relevant dimensions, the extent is unlimited. To compute the subspace clusters, the algorithm chooses randomly some medoid points and considers all possible hypercubes (the number is exponential in the dimension)

around each medoid point. For each hypercube, it then computes the support of the hypercube, i.e., the number of points which are very likely to be located within this hypercube. Based on the support and the number of relevant dimensions of the hypercube, the quality of the hypercube is computed. Finally, the hypercube with the highest quality is chosen as subspace cluster and the set of possible medoids is reduced. In this manner, the algorithm computes one subspace cluster after the other until the set of possible medoids is empty. To speed up computations, the algorithm deploys some pruning and sampling techniques.

Surprisingly, only a few theoretical results on clustering uncertain data have been obtained so far [17, 33]. Cormode and McGregor [17] were the first who theoretically investigated probabilistic clustering problems. Here, probabilistic means that the input is a set of probabilistic points, each formalized as a discrete probability distribution function which describes the possible locations of the points. Cormode and McGregor considered two variations of probabilistic clustering. In the first variation, called *unassigned clustering*, each point is assigned to the closest cluster center. In the second variation, called *assigned clustering*, each point is assigned to a fixed cluster center, no matter where it is actually located. Cormode and McGregor obtained a $(1 \pm \varepsilon)$-approximation for the unassigned Euclidean k-median problem and both the unassigned and assigned Euclidean k-means problem by using simple reductions to weighted deterministic clustering problems. For the assigned metric k-median problem, they achieved a constant factor approximation by first computing the 1-median of each probabilistic input point and then clustering the 1-medians. For the unassigned metric k-center problem, they proposed a bicriteria approximation algorithm which results in a constant factor approximation but uses $2k$ instead of k cluster centers.

In a follow-up work, Guha and Munagala [33] improved the last-mentioned result of Cormode and McGregor. More precisely, by using a reduction to a "truncated" version of a deterministic metric k-median problem, they obtained a constant factor approximation for both the unassigned and assigned metric k-center problem that preserves the number of allowed cluster centers k.

A survey of uncertain data mining and management applications can be found in [2].

5 Database Approaches to High-Dimensional Nearest Neighbor Searching

Nearest neighbor searching (NNS), also known as similarity searching or similarity retrieval, is a fundamental computational problem that has significant applications in many fields of computer science. In this section we summarize the most efficient methods for NNS that have been developed by the database community.

The problem in its general form is defined as follows. Let U be a set of elements and let d be a distance function that maps each pair of elements from U to some positive real number. Typically d is a metric distance function (i.e., it satisfies the triangle inequality) although this need not always be the case [45]. Given a set S of n elements from U, the goal is to find fast for any query element q in U the closest element to q among the elements of S, that is an element minimizing the distance $d(q, e)$ over all e in S. If besides the closest element we wish to report the k closest elements to q for $1 < k < n$ we have the so called *k-NN* problem.

In the following we will consider a somewhat restricted version of NNS, which is however the standard version studied in Computational Geometry, and to which many instances of NNS reduce. Let P be a set of n points in $\mathcal{R}^d$ and let $d(p, p')$ denote the Euclidean distance between any two points p and p'. We want to pre-

process P into an external memory data structure or *index* so that given any query point q in $\mathcal{R}^d$ we can efficiently report a *nearest neighbor (NN)* to q among the points in P. We say that a point p in P is a nearest neighbor of q if for any point $p' \in P$, it holds that $d(q,p) \leq d(q,p')$.

Since an exact solution to high-dimensional NNS requires heavy resources many techniques focus on the less demanding task of computing an approximate nearest neighbor. Given a parameter $c > 1$, a *c-approximate nearest neighbor (c-NN)* to a query q is a point p in P with $d(q,p) \leq c \cdot d(q,p')$ where p' is a NN to q. Hence, under approximation, the answer can be any point whose distance from q is at most c times larger than the distance between q and its NN.

There is a large literature on indexes for NNS [11, 59, 60]. The subsections below present three kinds of indexes, selected on the following criteria: they are the state-of-the-art in the field and they are efficient in high dimensions, they give provably correct results, and they represent different paradigms.

Some of these indexes are based on a standard data structure, the B^+-tree [16]. This is a variant of the B-tree [59, appendix A], where the internal nodes store just the keys, and the leaves are linked together as in a sorted doubly-linked list. It occupies $O(n/B)$ pages of space and has query time of $O(\log_B n)$ page accesses, where B is the page size. Predecessor/successor queries beginning at a leaf require at most one page access. B^+-trees are used to implement R-trees [60, sec.6.3], employed to store spatial objects more general than points. The latter were modified to define R^*-trees (cf. [7], [59, sec.2.1.5]), which are considered state-of-the-art among object-based hierarchical interior-based representations of spatial objects.

5.1 Clustering and the iDistance Method.

Jagadish et al. [39] proposed the iDistance method which offers a solution to the k-NN problem. The index is built as follows. First m points are chosen as *centers*, where m is a user-defined parameter. The authors study various schemes for selecting the centers which could depend or not on the data distribution. For instance, for clustered datasets they successfully use the cluster centers that the k-means algorithm computes. After picking the centers, set P is partitioned into m subsets P_j for $1 \leq j \leq m$ commonly by associating each point in P to its closest center. Each point p in P is assigned a one-dimensional real value v_p that is equal to its distance from its associated center. Finally these n values are stored in a B^+-tree [59] after appropriate translation so that for any j the values of the points in P_j and those of the points in P_{j+1} lie in consecutive intervals.

A query q is processed as follows. The search for the k-NNs starts with a ball b_r of radius r centered at q. The radius r is given a default value that is increased by a fixed amount whenever b_r is found to contain less than k points. The algorithm considers one by one each subset P_j. For each j it computes v_q, the distance of q from the center of P_j. By a simple descent in the B^+-tree, it locates the predecessor of q, that is the point p in P_j having the largest value v_p smaller than v_q, and it computes the actual distance $d(q,p)$. The algorithm maintains a set S of at most k points that are potential k-NNs to q. Depending on $d(q,p)$ and S, it checks whether S should be updated with p. After this the search continues similarly to check the points in P_j to the left of p.

To limit unnecessary calculations, some well-known but very effective pruning heuristics are used [59]: By the triangle inequality, for any p in P_j it holds that $|v_q - v_p| \leq d(q,p)$. If for some p it holds that $v_q - v_p > r$, then p and all the points left of p in P_j, which have values $\leq v_p$, lie outside b_r and thus they can be omitted. At some point the search towards left of q stops. Then the search continues to the right of q. Here the symmetric stop condition is $v_p - v_q > r$. Once the search on set P_j has been completed, q is located among the points in P_{j+1} and the same type

of bidirectional linear scan is executed on P_{j+1}. After visiting all m subsets, the algorithm checks if S contains k points inside ball b_r. If so, the search finishes and S is reported as the solution. Otherwise the radius r is increased, and the sets P_j are revisited with the search resuming from where it stopped in the last round.

Experiments showed that the iDistance method is faster by a factor from 2 to 6 compared to other indexes such as the M-tree [13] and the Omni-sequential [25] or the internal memory based BD-tree [50]. One drawback is that for uniform data distributions as the dimension increases the method becomes not competitive to linear scan. Poor query performance also occurs in the worst-case, when for example a large number of points lie at roughly the same distance from a center. On the positive side, if the centers are carefully selected iDistance gives good results for clustered datasets, it always returns the exact k-NNs, and it relies on the commonly used B^+-tree.

5.2 Rank Aggregation and the Medrank Method.

Fagin et al. [24] give a solution to NNS by relating it to *rank aggregation*. Rank aggregation is the following problem: Suppose that there are d voters and n candidates, and that each voter ranks all the n candidates. The goal is to determine based on these rankings which of the candidates should be the winner of this voting. The authors show how NNS can be reduced to a certain kind of rank aggregation. First using standard dimension reduction results [41, 38] (see Section 2), the query q and the points in P are projected down to $O(\log n/(c-1)^2)$ dimensions. Without ambiguity we denote the projected query, the projected point set and the reduced dimensions by q, P and d respectively as well. Let $q = (q_1, q_2, \ldots, q_d)$. The d coordinates play the role of the d voters and the n points that of the n candidates. For each i, q_i ranks the n points according to the distance of their ith-coordinate from q_i. Thus the point with the closest ith-coordinate to q_i (ties broken arbitrarily) is set first in the rank, the point with the second closest ith-coordinate second, and so on. The question is what the overall 'closeness' rank of a point p for q should be based on all the d independent coordinate rankings. The authors propose as a good overall ranking the *medrank*(p) which is defined as the median of all the d coordinate rankings for p. The winner of the voting, which is the estimated near neighbor to q, is then the point having the minimum medrank over all points in P.

The estimated near neighbor is not a c-NN in all cases. However if the rank for a point with respect to q_i is replaced by a score that takes into account the absolute distance from q_i and medrank is replaced by *medscore*, the median of these scores, then with high probability the point that minimizes medscore is guaranteed to be a c-NN [4].

The point of minimum medrank is computed by the following algorithm. At preprocessing d doubly-linked lists are formed, where the i-th list contains all the i-coordinates from all the points in P in increasing order. With each coordinate a pointer to the corresponding point is also stored. For large n the lists can be stored with B^+-trees. When a query q arrives, q is located in each of the d lists. Then a bidirectional scan is initiated from q's position in each list. At each step the scan progresses in the direction and in the list that has the next point with the smallest rank among all the $2d$ possible current points. Thus each step requires $O(1)$ time and at each step a new point is examined. The first point that appears more than $d/2$ times can be shown to be the point with the minimum medrank. If this process continues the points with the top-k medranks are similarly found. Interestingly the above algorithm despite its simplicity is *instance optimal*. This means that it checks at most a constant factor more points than any algorithm does which also computes the point of minimum medrank by sequential or random accesses in the d sorted lists.

According to the experimental results the Medrank method for NNS is practical for dimensions as high as 200. The dimension reduction is important for this. Moreover even though in theory the approximation factor is not guaranteed, in most cases it turns out to be small.

5.3 Locality Sensitive Hashing and the LSB-Tree Method.

The locality sensitive B-tree (LSB-tree) by Tao et al. [62] is based on *locality sensitive hashing (LSH)* [18, 38]. It improves significantly upon an earlier external memory implementation of LSH for NNS by Gionis et al. [29]. The index is constructed as follows. Each point p in P is projected to a point p' in $d' = O(\log(dn/B))$ dimensions. Specifically, for $1 \leq i \leq d'$ the i-th coordinate of p', denoted by p'_i, is given by $\vec{a}_i \cdot \vec{p} + b_i$ where $\vec{a}_i$ is a random d-dimensional vector and b_i is a random number chosen uniformly from a interval of fixed length. The space region occupied by the n projected points is then partitioned into hypercube grid-cells of equal size (we simply call them cells) so that there are $(2^s)^{d'}$ cells in total, where s is a large enough integer constant. Thus each projected point p' lies in a cell which can be identified by d' s-bit coordinates with each coordinate giving the order of the cell in the corresponding dimension.

In [29, 38] the locality sensitive properties of the projection ensure that a query q is likely to be mapped in the same cell with one of its c-NNs. In this method a different approach is followed in the sense that either the cell containing q' or a nearby nonempty cell is likely to contain a c-NN to q. To locate efficiently neighboring cells the well-known Z-order curve is used [27]. Let c_p be the cell containing a point p in the projected space. Let $b_i^{(j)}$ denote the j-th bit of the i-th coordinate of c_p. Then the Z-order value assigned to p and c_p is the binary number

$$b_1^{(s)} b_2^{(s)} \dots b_{d'}^{(s)} b_1^{(s-1)} b_2^{(s-1)} \dots b_{d'}^{(s-1)} \dots b_1^{(1)} b_2^{(1)} \dots b_{d'}^{(1)}.$$

The Z-order values impose a linear ordering on all the n points and their associated cells. The benefit of this ordering is that neighboring points and cells are likely to receive close Z-order values. Finally these n values are stored for fast location in B^+-tree which gives the LSB-tree. Along the lines of [29], the same construction is repeated $\ell = O(\sqrt{dn/B})$ times (each time with different random vectors $\vec{a}_i$ and values b_i) which guarantees with constant probability that a 4-NN is returned. The set of these ℓ LSB-trees is called *LSB-forest*.

The query processing starts by computing the Z-order of q (more accurately of its projection q' but for simplicity we keep the same notation) and locating the leaf that contains it in each of the ℓ LSB-trees. Then a bidirectional scan is initiated much like in the Medrank method (see previous subsection). In this method however the point that is examined at each step is the one maximizing the *length of the longest common prefix (LLCP)* among the 2ℓ current points in the ℓ trees. The LLCP for a point p is computed between the binary strings of the Z-order of p and that of the query point q. Roughly speaking, the larger the value of LLCP, the closer the two points are. Moreover, as the search continues the LLCP values of points examined are decreasing. The scanning stops when either $O(\ell B/d)$ points have been checked or when there is proof based on the current LLCP value that no subsequent point can be much closer than the best NN found so far. The method can be extended to compute also the approximate k-NNs to q. The space of the LSB-forest is $O((dn/B)^{3/2})$ and the query time is $O(\sqrt{dn/B} \cdot \log_B n)$, which is sublinear. Compared to the previous LSH scheme [29], the LSB-forest bounds have no dependence on the largest integer coordinate T of the points in P and in fact the space bound is better by a factor $O(\log(dT))$.

Extensive experimental results were provided comparing the LSB-tree and the LSB-forest with the iDistance method, the Medrank method and the LSH methods in [29]. LSB-tree offers good quality results with faster query times and about the same or less space in comparison to all the other methods in dimensions up to 100. LSB-forest achieves a theoretical approximation error of 4 and in practice of at most 1.5 at the cost of increased space.

5.4 Discussion

We juxtapose the previous presentation with a brief mention of the most relevant approaches and software coming from the Computational Geometry community, which can efficiently treat high-dimensional data. These methods put the emphasis on robustness and theoretical guarantees, and assume the space dimension is constant when analyzing space and time complexity; nonetheless, in practice space dimension can be quite large.

An important instance is the work of Arya et al. on approximate nearest neighbors (ANN), e.g. [6]. This approach, implemented on Balanced Box-Decomposition (BBD) trees [6], led to `ANN`, one of the state-of-the-art software packages [50]. Another successful package is `FLANN` (Fast Library for Approximate Nearest Neighbor), which contains a collection of algorithms and data-structures among which the software automatically chooses the most appropriate for the input dataset, while optimizing the parameters [51].

Another geometric approach for ANN searching relies on Approximate Voronoi Diagrams, which are shown to establish a tradeoff between the space complexity of the data structure and the query time it supports [5]. They are implemented on a hierarchical quadtree-based subdivision of space into cells, each storing a number of representative points, such that for any query point lying in the cell, at least one of the representatives is an approximate nearest neighbor.

6 Summary and Conclusion

The fields of Computational Geometry, Computational Learning, and Databases have an overlap where they study high-dimensional data. We have seen that many basic tools are in common to these areas. Research in Databases focuses of running systems, and many papers report impressive experimental results. On the other hand, Computational Geometry has a strong emphasis on getting a thorough understanding, trying to explain what works and what can be proved. In this respect, the approaches in the different fields are complementary, but they can cross-fertilize and learn from each other by studying their various approaches and techniques.

Acknowledgement. Partially supported by the IST Programme of the EU (FET Open) Project under Contract No IST-25582 – (CGL - Computational Geometric Learning)

References

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66:671–687, 2003.

[2] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.*, 21(5):609–623, 2009.

[3] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[4] A. Andoni, R. Fagin, R. Kumar, M. Patrascu, and D. Sivakumar. Corrigendum to "Efficient similarity search and classification via rank aggregation" by R. Fagin, R. Kumar and D. Sivakumar (SIGMOD 2003). In *Proc. ACM SIGMOD Intern. Conf. Management of Data*, pages 1375–1376, 2008.

[5] S. Arya, T. Malamatos, and D.M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57:1–54, 2009.

[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, 1998.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Intern. Conf. Management of Data*, pages 322–331, Atlantic City, NJ, 1990.

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[9] M. Belkin and P. Niyogi. Towards a theoretical foundation for Laplacian-based manifold methods. *Journal of Computer and System Sciences*, 74(8):1289–1308, 2008.

[10] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.

[11] C. Böhm, S. Berchtold, and D.A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33:322–373, 2001.

[12] Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng. Uncertain data mining: An example in clustering location data. In *Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2006), volume 3918 of Lecture Notes in Computer Science*, pages 199–204. Springer, 2006.

[13] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23rd Intern. Conf. Very Large Data Bases*, pages 426–435, 1997.

[14] R. R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proc. of Nat. Acad. Sci.*, 102:7426–7431, 2005.

[15] R. R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Multiscale methods. *Proc. of Nat. Acad. Sci.*, 102:7432–7437, 2005.

[16] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[17] Graham Cormode and Andrew McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2008)*, pages 191–200, 2008.

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SoCG*, pages 253–262, 2004.

[19] V. de Silva, J.C. Langford, and J.B. Tenenbaum. Graph approximations to geodesics on embedded manifolds. Technical report, Pomona College, 2000.

[20] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*. MIT Press, Cambridge, MA, 2003.

[21] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[22] D. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 1996)*, pages 226–231, 1996.

[24] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proc. ACM SIGMOD Intern. Conf. Management of Data*, pages 301–312, 2003.

[25] R. Filho, A. Traina, C. Traina, and C. Faloutsos. Similarity search without tears: The OMNI-family of all-purpose access methods. In *Proc. 17th Intern. Conf. Data Engineering*, pages 623–632, 2001.

[26] E. W. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. *Biometrics*, 21:768–780, 1965.

[27] V. Gaede and O. Guenther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[28] J. Giesen and U. Wagner. Shape dimension and intrinsic metric from samples of manifolds with high co-dimension. In *Proc. 19th Annual Symp. Computational Geometry*, pages 329–337, 2003.

[29] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In ACM, editor, *Proc. Conf. VLDB*, pages 301–312, 1999.

[30] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[31] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):515–528, January/February 2003.

[32] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS '00)*, pages 359–366. IEEE Computer Society, 2000.

[33] Sudipto Guha and Kamesh Munagala. Exceeding expectations and clustering uncertain data. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2009)*, pages 269–278, 2009.

[34] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001.

[35] Stephan Günnemann, Hardy Kremer, and Thomas Seidl. Subspace clustering for uncertain data. In *Proceedings of the SIAM International Conference on Data Mining (SDM 2010)*, pages 385–396, 2010.

[36] Matthias Hein and Markus Maier. Manifold denoising. In *Neural Information Processing Systems*, pages 561–568, 2006.

[37] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):145–153, 1979.

[38] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. STOC*, pages 604–613, 1998.

[39] H.V. Jagadish, B. Chin Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Systems*, 30(2):364–397, 2005.

[40] Leonard Kaufman and Peter J. Rousueeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[41] J.M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. 29th Annual ACM Symp. Theory of Computing*, pages 599–608, New York, 1997. ACM.

[42] Hans-Peter Kriegel and Martin Pfeifle. Density-based clustering of uncertain data. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005)*, pages 672–677, 2005.

[43] Hans-Peter Kriegel and Martin Pfeifle. Hierarchical density-based clustering of uncertain data. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 689–692, 2005.

[44] S. Lafon and A.B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning and data set parameterization. *IEEE PAMI*, 28(9):1393–1403, 2006.

[45] Y. Lifshits and S. Zhang. Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design. In *Proc. SODA*, pages 318–326, 2009.

[46] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[47] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[48] Jiří Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag New York, 2002.

[49] F. Memoli and G. Sapiro. Distance functions and geodesics on point clouds, 2005.

[50] D.M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. In *Proc. 2nd Center for Geometric Computing Workshop on Computational Geometry*, 1997.

[51] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. Intern. Conf. Computer Vision Theory & Appl. (VISSAPP)*, pages 331–340. INSTICC Press, 2009.

[52] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 144–155, 1994.

[53] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip. Efficient clustering of uncertain data. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006)*, pages 436–445, 2006.

[54] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha. Streaming-data algorithms for high-quality clustering. In *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*, pages 685–696. IEEE Computer Society, 2002.

[55] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[56] KV Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *ACM SIGMOD Record*, 27(2):166–176, 1998.

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

[58] S. T. Roweis and L. K. Saul. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

[59] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. MorganKaufmann, 2006.

[60] H. Samet. Multidimensional data structures for spatial applications. In M.J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*, chapter 6. CRC Press, Boca Raton, Florida, 2010.

[61] Shokri Z. Selim and Mohamed A. Ismail. k-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 6(1):81–87, January 1984.

[62] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35:1–46, 2010.

[63] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[64] Alex Thomasian, Vittorio Castelli, and Chung sheng Li. RCSVD: Recursive clustering with singular value decomposition for dimension reduction in content-based retrieval of large image/video databases. Technical Report RC 20704, IBM Research Division, T. J. Watson Research Center, 1997.

[65] L.J.P. van der Maaten, E.O. Postma, and H.J. van den Herik. Dimensionality reduction: a comparative review. Technical Report TiCC-TR 2009-005, Tilburg University, 2009.

[66] Kilian Q. Weinberger and Lawrence K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 988–995, 2004.

[67] K.Q. Weinberger and L.K. Saul. An introduction to nonlinear dimensionality reduction by maximum variance unfolding. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1683–1686. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[68] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[69] Huajie Xu and Guohui Li. Density-based probabilistic clustering of uncertain data. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 04*, CSSE '08, pages 474–477, Washington, DC, USA, 2008. IEEE Computer Society.

[70] Hao Zhang, Oliver van Kaick, and Ramsay Dyer. Spectral mesh processing. *Computer Graphics Forum*, 29(6):1865–1894, 2010.

[71] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, June 1997.