



Hierarchical data organization, clustering and denoising via localized diffusion folders

Gil David^{a,*}, Amir Averbuch^b

^a Department of Mathematics, Program in Applied Mathematics, Yale University, New Haven, CT 06510, USA

^b School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel

ARTICLE INFO

Article history:

Received 18 October 2010

Revised 2 September 2011

Accepted 4 September 2011

Available online 16 September 2011

Communicated by Charles K. Chui

Keywords:

Hierarchical clustering

Diffusion geometry

Spectral graph

ABSTRACT

Data clustering is a common technique for data analysis. It is used in many fields including machine learning, data mining, customer segmentation, trend analysis, pattern recognition and image analysis. The proposed Localized Diffusion Folders (LDF) methodology, whose localized folders are called diffusion folders (DF), introduces consistency criteria for hierarchical folder organization, clustering and classification of high-dimensional datasets. The DF are multi-level data partitioning into local neighborhoods that are generated by several random selections of data points and DF in a diffusion graph and by redefining local diffusion distances between them. This multi-level partitioning defines an improved localized geometry for the data and a localized Markov transition matrix that is used for the next time step in the advancement of the hierarchical diffusion process. The result of this clustering method is a bottom-up hierarchical data organization where each level in the hierarchy contains LDF of DF from the lower levels. This methodology preserves the local neighborhood of each point while eliminating noisy spurious connections between points and areas in the data affinities graph. One of our goals in this paper is to illustrate the impact of the initial affinities selection on data graphs definition and on the robustness of the hierarchical data organization. This process is similar to filter banks selection for signals denoising. The performance of the algorithms is demonstrated on real data and it is compared to existing methods. The proposed solution is generic since it fits a large number of related problems where the source datasets contain high-dimensional data.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

Many recent graph based methodologies for data organization and analysis are based on a given affinity between data points provided by a kernel or a metric. This approach, which only compares two data points at a time, introduces spurious connections between points that look as noisy data. We denoise and reinforce the affinities between points by defining them to be between data folders. By using a randomized collection of folders, we build consistent data affinities. A related idea has been recently introduced in [1] for manifold learning where point neighborhoods are matched through rotations or local isometrics, leading to powerful data denoising tools. Our approach does not require to know the underlying manifold structure since it defines a preponderance of links based affinity. We use this affinity to generate hierarchical data partitioning. This partitioning provides organization and data clusters whenever they exist. We begin with the Diffusion Maps (DM) framework [2] with its inherent diffusion distances that provide a method for finding meaningful geometric structures in datasets. In most cases, the dataset contains high-dimensional data points in \mathbb{R}^n . DM constructs coordinates that parame-

* Corresponding author. Fax: +1 203 432 4345.

E-mail address: gil.david@yale.edu (G. David).

terize the dataset and the diffusion distance provides a local preserving metric for this data. A non-linear dimensionality reduction, which reveals global geometric information, is constructed by local overlapping structures. Let $\Gamma = \{x_1, \dots, x_m\}$ be a set of points in \mathbb{R}^n and μ is the distribution of the points on Γ . We construct the graph $G(V, E)$, $|V| = m$, $|E| \ll m^2$, on Γ in order to study the intrinsic geometry of this set. A weight function $W_\epsilon \triangleq w_\epsilon(x_i, x_j)$ is introduced. It measures the pairwise similarity between the points. For all $x_i, x_j \in \Gamma$, the weight function has the following properties: symmetry: $w_\epsilon(x_i, x_j) = w_\epsilon(x_j, x_i)$, non-negativity: $w_\epsilon(x_i, x_j) \geq 0$ and positive semi-definite: for all real-valued bounded functions f defined on Γ , $\int_\Gamma \int_\Gamma w_\epsilon(x_i, x_j) f(x_i) f(x_j) d\mu(x_i) d\mu(x_j) \geq 0$. A common choice for W_ϵ is $w_\epsilon(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{\epsilon}}$.

The non-negativity property of W_ϵ allows to normalize it into a Markov transition matrix P where the states of the corresponding Markov process are the data points. This enables one to analyze Γ as a random walk. The construction of P is known as the *normalized graph Laplacian* [3].

Formally, $P = \{p(x_i, x_j)\}_{i,j=1,\dots,m}$ is constructed as $p(x_i, x_j) = \frac{w_\epsilon(x_i, x_j)}{d(x_i)}$ where $d(x_i) = \int_\Gamma w_\epsilon(x_i, x_j) d\mu(x_j)$ is the degree of x_i . P is a Markov matrix since the sum of each row in P is 1 and $p(x_i, x_j) \geq 0$. Thus, $p(x_i, x_j)$ can be viewed as the probability to move from x_i to x_j in *one* time step. By raising this quantity to a power t (advance in time), this influence is propagated to nodes in the neighborhood of x_i and x_j and the result is the probability for this move in t time steps. We denote this probability by $p_t(x_i, x_j)$. These probabilities measure the connectivities among the points within the graph. The parameter t controls the neighborhood scale and ϵ the local neighborhood.

Construction of $\tilde{p}(x_i, x_j) = \frac{\sqrt{d(x_i)}}{\sqrt{d(x_j)}} p(x_i, x_j)$, which is a symmetric and positive semi-definite kernel, leads to the eigen-decomposition $\tilde{p}(x_i, x_j) = \sum_{k \geq 0}^{m-1} \lambda_k v_k(x_i) v_k(x_j)$ where λ_k and v_k , $k = 0, \dots, m$, are the eigenvalues and the eigenvectors of \tilde{p} , respectively. A similar eigen-decomposition is obtained by $\tilde{p}_t(x_i, x_j) = \sum_{k \geq 0}^m \lambda_k^t v_k(x_i) v_k(x_j)$ after advancing t times on the graph. Here \tilde{p}_t is the probability of the transition from x_i to x_j in t time steps.

A fast decay of $\{\lambda_k\}$ can be achieved by an appropriate choice of ϵ . Thus, only a few terms are needed in the sum above to achieve a given relative cover $\delta > 0$. Let $\eta(\delta)$ be the number of retained terms. A family of DM [2], denoted by $\Phi_t(x) \in \mathbb{R}^m$, $m \in \mathbb{N}$, is given by $\Phi_t(x) = (\lambda_0^t v_0(x), \lambda_1^t v_1(x), \dots, \lambda_{\eta(\delta)-1}^t v_{\eta(\delta)-1}(x))^T$, where the map $\Phi_t : \Gamma \rightarrow \mathbb{R}^m$ embeds the dataset Γ into a Euclidean space \mathbb{R}^m . The *diffusion distance* is defined as $D_t^2(x_i, x_j) = \sum_{k \geq 0} (\tilde{p}_t(x_i, x_k) - \tilde{p}_t(x_k, x_j))^2$. This formulation is derived from the random walk distance in Potential Theory. It is shown in [2] that the diffusion distance can be expressed in terms of the right eigenvectors of P : $D_t^2(x_i, x_j) = \sum_{k \geq 0} \lambda_k^{2t} (v_k(x_i) - v_k(x_j))^2$. It follows that in order to compute the diffusion distance, one can simply use the eigenvectors of \tilde{P} . Moreover, this facilitates the embedding of the original points in a Euclidean space $\mathbb{R}^{\eta(\delta)-1}$ by $\Xi_t : x_i \rightarrow (\lambda_0^t v_0(x_i), \lambda_1^t v_1(x_i), \lambda_2^t v_2(x_i), \dots, \lambda_{\eta(\delta)}^t v_{\eta(\delta)}(x_i))$. This also provides coordinates on the embedded space. Essentially, $\eta(\delta) \ll m$ due to the fast spectral decay of the spectrum of P . Furthermore, $\eta(\delta)$ depends only on the primary intrinsic variability of the data as captured by the random walk and not on the original dimensionality of the data. This data-driven method parameterizes any set of points provided the availability of similarity matrix W_ϵ between data points.

As described in the brief DM overview above, P is the affinity matrix of the dataset and it is used to find the diffusion distances between data points. This distance metric can be used to cluster the data points according to the propagation of the diffusion distances that are controlled by t . In addition, it can be used to construct a bottom-up data hierarchical clustering. For $t = 1$, the affinity matrix reflects local and direct connections between adjacent data points. The resulting clusters preserve the local neighborhood of each point. These clusters are the bottom level of the hierarchy. By raising t (advancing in time), the affinity matrix is changed accordingly and it may reflect indirect connections between data points in the graph. The diffusion distance between data points in the graph accounts for all possible paths of length t between these data points in a given time step. The more we advance in time, the more we may increase indirect global connections that violate the sought after locality. Therefore, by raising t we construct the upper levels of the clustering hierarchy. In each advance in time, it is possible to merge more and more lower-level clusters since there are more and more new paths between them. The resulting clusters reflect global neighborhood of each data point that is highly affected by the advances of the parameter t .

The major risk in this global approach is that by increasing t , noise, which is classified as connections between points that are not related in the affinity matrix, increases as well. Moreover, clustering errors in the lower levels of the hierarchy will diffuse to the upper levels of the hierarchy and hence will significantly affect the correctness of the upper levels clustering. As a result, some areas in the graph, which assumed to be separated, will be connected by the new noise-result and error-result paths. Thus, erroneous clusters will be generated. This type of noise significantly affects the diffusion process and eventually the resulting clusters will not reflect the correct relations among data points. Although these clusters consist of data points that are adjacent according to their diffusion distances, the connections among these points in each cluster can be too global and too loose to generate inaccurate clusters.

In this paper, we present a hierarchical clustering method of high-dimensional data via what we call *localized diffusion folders* (LDF) [4]. This method overcomes the problems that were described above. It is based on the key idea that clustering of data points should be achieved by utilizing the local geometry of the data and the local neighborhood of each data point and by constructing a new local geometry every advance in time. The new geometry is constructed according to the local connections and according to diffusion distances in previous time steps. This way, as we advance in time, the geometry from the induced affinity reflects better the data locality while ‘‘affinity’’ noise in the new localized matrix decreases and

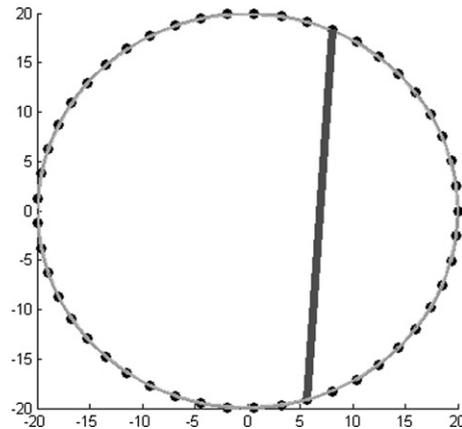


Fig. 1.1. Circle of points where the lines correspond to the affinity between every adjacent points and the anomalous pair of points. The strong line, which connects far points (not neighbors), is classified as noise.

the accuracy of the resulting clusters increases. LDF is introduced to achieve local geometry that is preserved along the hierarchical construction. The LDF framework provides a multi-level partitioning (similar to Voronoi diagrams in diffusion metric) of data into local neighborhoods that are initiated by several random selections of data points or folders of data points in the diffusion graph and by defining local diffusion distances between them. Since every different selection of initial points yields a different set of folders, it is crucial to repeat this selection process several times. The multiple system of folders, which we get at the end of this random selection process, defines a new affinity and this defines a new geometry on the graph. This localized affinity is a result of what we call “shake n bake” process: first, we “shake” the multiple Voronoi diagrams together in order to get rid of the noise in the original affinity. Then, we “bake” a new cleaner affinity that is based on the actual geometry of the data while eliminating rare connections between data points. This affinity is more accurate than the original affinity since instead of defining a general affinity on the graph, we let the data define its localized affinity on the graph. In every time step, this multi-level partitioning defines a new localized geometry of the data and a new localized affinity matrix that is used by the next time step. In every time step, we use the localized geometry and the localized folders that were generated in the previous time step to define the localized affinity between folders. The affinity between two folders is defined by the localized diffusion distance metric between data points in the two folders. In order to define this distance between the folders, we construct a local sub-matrix that contains only the affinity of the data points (or folders) of the two folders. This sub-matrix is raised to the power of the current time step (according to the current level in the hierarchy) and then it is used to find the localized diffusion distance between the two folders. The result of this clustering method is a bottom-up hierarchical data clustering where each level in the hierarchy contains DF of folders from the lower levels. Each level in the hierarchy defines a new localized affinity (geometry) that is dynamically constructed and it is used by the upper level. This methodology preserves the local neighborhood of each data point while eliminating the noisy connections between distinct data points and areas in the graph.

1.1. Illustrative example

The following example illustrates the need to remove noisy connections to guarantee the robustness and the correctness of the hierarchal data organization via iterative process that reveals and processes the underlying local geometry.

Fig. 1.1 shows a circle of points. Each point on the circle is connected to its left and right immediate neighboring points with a normalized weight of 0.183 and connected to the its left and right second neighboring points with a normalized weight of 0.1. The lines between these points correspond to their affinities. An artificial connection between two distinct points (the wide line that crosses the circle) with a normalized weight of 0.183 was added to Fig. 1.1. This connection is classified as noise.

We used the original affinity to construct and refine the localized affinity matrix, as described in details in Section 3. As the computation in Section 3.2 Step 8 shows, 55 Voronoi systems are required to construct the localized affinity matrix.

Fig. 1.2 shows the LDF affinity using different number of Voronoi systems. When 10 systems are used, several original connections are lost. When the number of fused systems increases, more original connections are reconstructed and the affinity is stabilized after 60 Voronoi systems are fused. In addition, the noisy line in the original affinity was eliminated in each of the LDF affinities. At the end of this process, each point on the circle was connected to the first point on its right and left sides with an average normalized weight of 0.2 and connected to the second point on its right and left sides on the circle with an average normalized weight of 0.13. These are exactly the normalized weights that we get when repeating this process on a circle that does not contain noisy lines at all.

Fig. 1.3 shows how the original diffusion algorithm handles this affinity. First, we constructed the Markov matrix with $t = 1$ as it is described in this section. Then, we eliminated loose connections that were smaller than a threshold (the

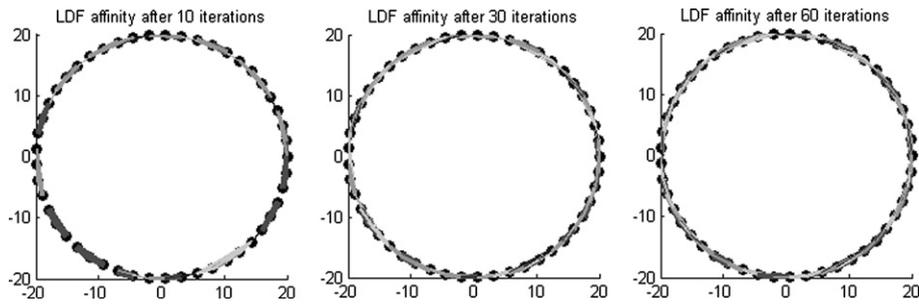


Fig. 1.2. LDF affinities after different number of iterations. Left: After 10 iterations. Middle: After 30 iterations. Right: After 60 iterations.

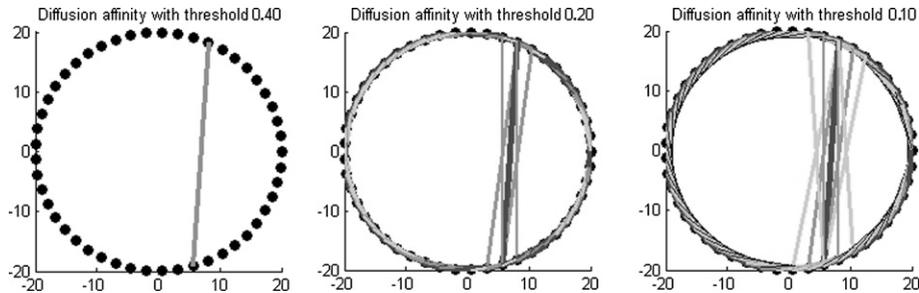


Fig. 1.3. Diffusion affinity with different thresholds. Left: Threshold = 0.4. Middle: Threshold = 0.2. Right: Threshold = 0.1. We see that noisy connections are still present.

probability to move from one point to another was smaller than the threshold). We repeated this process for $t = 1, \dots, 5$ with different thresholds. As we can see, when the threshold is too big, all the original connections are not reconstructed while the noisy connection is not eliminated. When the threshold is too small, many artificial connections are generated around the original noisy connection and as a result the affinity is much more noisy than the original affinity.

One of the goals in this paper is to show the impact of the initial affinities selection on data graphs definition and on the robustness of hierarchical data organization. This process is similar to filter banks selection for signals denoising. In particular, we will cover the following topics: constructing a Gaussian kernel with an adaptive scale control, denoising a given affinity matrix, defining a robust affinity matrix, fusing multiple systems of affinities and constructing an hierarchy of affinities.

The paper has the following structure. Section 2 presents related work. In Section 3, we present in details the methodology of the localized diffusion folders. Section 4 presents experimental results.

2. Related work

A multi-scale diffusion based organization of data is introduced in [5]. We follow a similar path in building partition trees. These trees lead to clustering whenever the linkage between a folder and its parent skips a level.

Many clustering algorithms have been proposed over the years. Most of them cluster numerical data, where the data consists of numeric attributes whose values are represented by continuous variables. Finding similarity between numeric objects usually relies on common distance measures such as Euclidean, Manhattan, Minkowski, Mahalanobis distances to name some. A comprehensive survey of several clustering algorithms is given in [6].

k -means [7] is a popular clustering algorithm. It was designed to cluster numerical data in which each cluster has a center called the mean. The k -means algorithm is classified as either partitioner or non-hierarchical clustering method. The performance of k -means is highly dependent on the initialization of the centers which is a drawback.

An agglomerative hierarchical algorithm is proposed in BIRCH [8]. It is used for clustering large numerical datasets in Euclidean spaces. BIRCH performs well when the clusters have identical sizes and their shapes are either convex or spherical. However, it is affected by the input order of the data and it may not perform well when clusters have either different sizes or non-spherical shapes.

CURE [9] is another method for clustering numerical datasets using hierarchical agglomerative algorithm. CURE can identify non-spherical shapes in large databases that have different sizes. It uses a combination of random sampling and partitioning in order to process large databases. Therefore, it is affected by the random sampler performance.

A density-based clustering algorithm is proposed in DBSCAN [10]. This method is used to discover arbitrarily shaped clusters. DBSCAN is sensitive to its parameters, which in turn, are difficult to determine. Furthermore, DBSCAN does not perform any pre-clustering and it executed directly on the entire database. As a result, DBSCAN can incur substantial I/O costs in processing large databases.

DIANA [11] is a divisive hierarchical algorithm that applies to all datasets that can be clustered by hierarchical agglomerative algorithms. Since the algorithm uses the largest dissimilarity between two objects in a cluster such as the diameter of the cluster, it is sensitive to outliers.

Several clustering algorithms for categorical data have been proposed in recent years such as [12]. The k -modes algorithm [13] emerged from the k -means algorithm and it is designed to cluster categorical datasets. The main idea of the k -modes algorithm is to specify the number of clusters and then to select k initial modes, followed by allocating every object to its nearest mode. The algorithm minimizes the dissimilarity of the objects in a cluster with respect to its mode.

The inter-attribute and intra-attribute summaries of a database are constructed in CACTUS [14]. Then, a graph, called the similarity graph, is defined according to these summaries. Finally, the clusters are found with respect to these graphs.

More recently, some clustering that are based on kernel methods have been proposed. A clustering method, which uses support vector machine, is given in [15]. Data points are mapped by a Gaussian kernel to a high-dimensional feature space, where it searches the minimal enclosing sphere. When this sphere is mapped back to the data space it can be separated into several components where each encloses a separate cluster.

A method for unsupervised partitioning of a data sample, which estimates the possible number of inherent clusters that generate the data, is described in [16]. It exploits the notion that performing a non-linear data transformation into some high-dimensional feature space increases the probability for linear separability between the patterns within the transformed space. Therefore, it simplifies the associated data structure. It shows that the eigenvectors of a kernel matrix, which define an implicit mapping, provide means to estimate the number of clusters inherent within the data. A computational iterative procedure is presented for the subsequent feature space that partitions the data.

A kernel clustering scheme, which is based on k -means for large datasets, is proposed in [17]. It introduces a clustering scheme which changes the clustering order from a sequence of samples to a sequence of kernels. It employs a disk-based strategy to control the data.

Another kernel clustering scheme, which is based on k -means, is proposed in [18]. It uses a kernel function that is based on Hamming distance to embed categorical data in a constructed feature space where clustering takes place.

Our proposed method is different from the above clustering methods in several aspects. First, it provides a multi-level (multi-scale) hierarchical clustering of the data where each level t in the hierarchy expresses the diffusion geometry of the data after t time steps. A level in our hierarchy means an advance in time in the diffusion graph. Moreover, our method is unique since it constructs a new localized geometry for the data at each time step. This enables a more accurate representation of the geometry of the data and, as a result, it provides a better clustering. Last, in every level in the hierarchy, our method removes from the data the noisy data points, which are the global loose connections between data points.

3. Localized Diffusion Folders (LDF) methodology and algorithm

The basic operation, which is based on repeated application of the LDF methodology to two consecutive scales, has two steps. In the first step, the bottom level of the hierarchy is constructed as follows: First, the data points are partitioned into diffusion folders (DF). Then, multiple systems of DF are constructed. Next, a localized affinity, which defines the “purified” geometry (we call it “shake n bake” process), is applied. Last, the data is partitioned into LDF.

In the second step, the upper levels of the hierarchy are constructed as follows: First, the localized affinity between the lower level DF is constructed. Then, multiple systems of DF of folders are constructed. Next, a localized affinity that defines the “purified” geometry (“shake n bake” process) is applied. Last, the DF are partitioned into LDF. This two-scale process is repeated till the root of the hierarchy is reached.

We begin with data points and continue bottom up with DF. In this section, we describe the LDF algorithm in details. Section 3.1 outlines the methodology. Section 3.2 describes the construction of the bottom level in the hierarchy ($t = 1$). Section 3.3 describes the construction of the upper levels in the hierarchy ($t > 1$).

3.1. Outline of the LDF methodology

In order to provide some intuition for the proposed construction, we use some simple examples to illustrate the proposed construction. The input to the algorithm is a set of data points as shown in Fig. 3.1(a). The first step constructs the Markov transition affinity matrix. This matrix defines the pairwise diffusion distances between every point in the graph to any other point in the graph. It is illustrated in Fig. 3.1(b).

Then, the DF are constructed using this affinity matrix. Initial partitioning of the data points into non-overlapping DF according to initial random selections of data points takes place.

Fig. 3.2(a) illustrates a system of non-overlapping DF. The bold points are the initial random selections. Since the resulting diffusion folders depend on the selection of the initial data points, we repeat this process several times with different random selections of the initial points. Each process yields a different system of DF. Fig. 3.2(b) shows two different systems of non-overlapping DF with different initial random selections.

We repeat this process several times according to the size of the dataset. The result of this process is a set of LDF systems where each system depends on the initial selection of the random data points. Fig. 3.3(a) shows this set of systems. Now, we have different sets of DF where each data point can reside in a different DF in each system. In other words, for

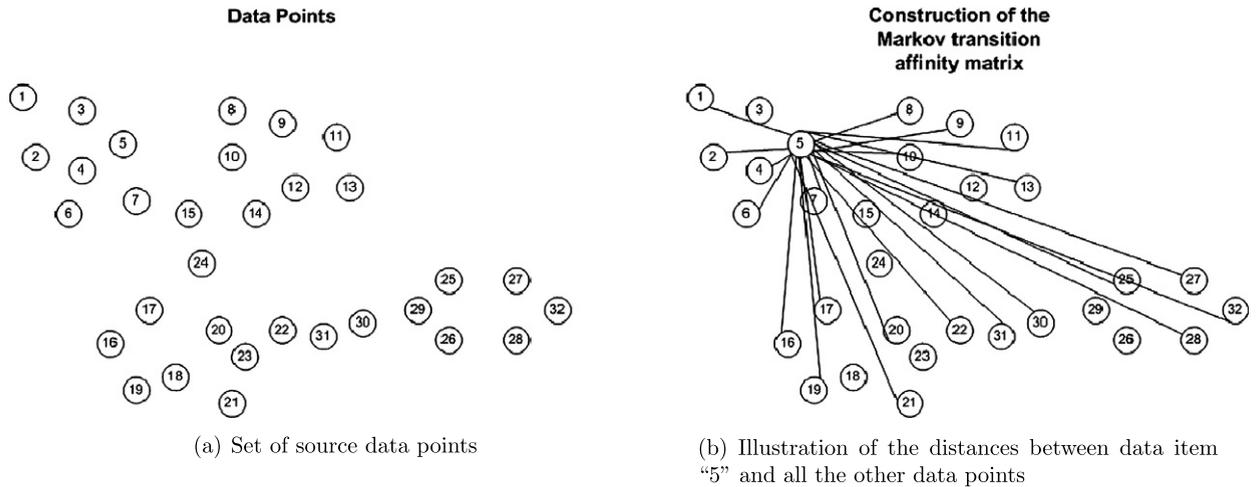


Fig. 3.1. First step ($t = 1$): Construction of the affinity matrix, which defines the pairwise diffusion distances between the data points.

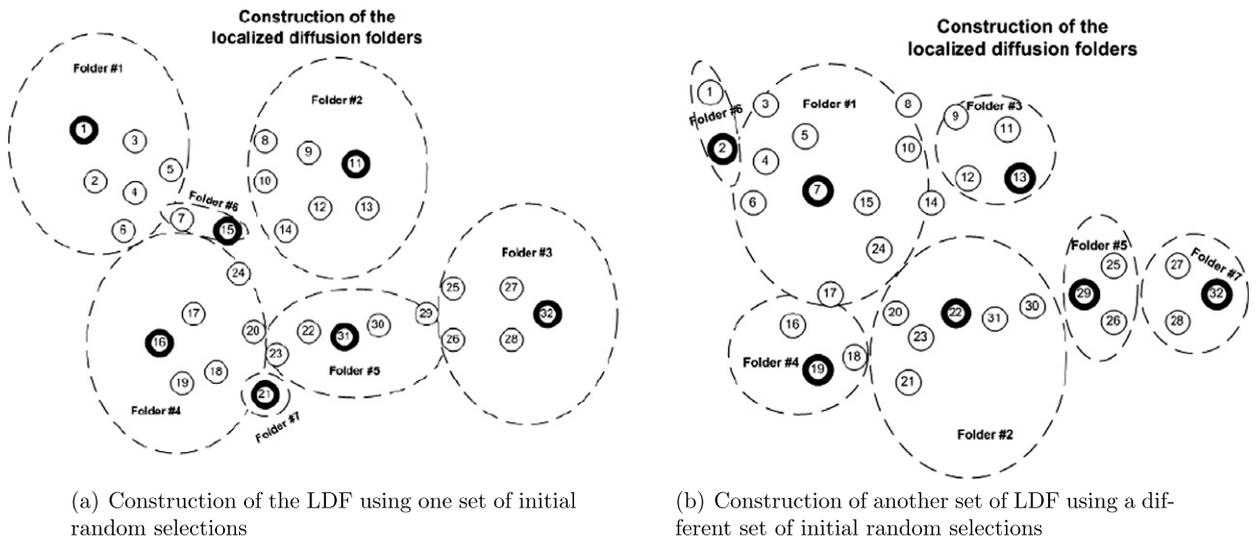


Fig. 3.2. Second step ($t = 1$): Construction of multiple LDF systems, where each system is initialized with a different set of random selections.

every set of DF, each data point can be associated with a different neighborhood. We construct the new localized affinity matrix by fusing multiple systems from different DF. For each data point, its distance to all other data points is defined as the relative probability to reside in the same neighborhood in the multiple systems of DF. This way, we use different sets of LDF to define a new affinity between data points. This affinity is more accurate than the original affinity since it reflects the actual neighborhoods of each data point. It also reduces the noise by eliminating rare connections. Fig. 3.3(b) shows the construction of the localized affinity matrix by fusing multiple systems. Once we have this improved affinity, the super-system of LDF is constructed using this localized affinity matrix. Fig. 3.3(c) shows this constructed system.

At this point, we constructed the bottom level of the hierarchy. This level contains DF of data points.

In order to improve the clustering, we build the next (upper) level in the hierarchy. This level reflects data clustering after advancing one time step. This level constructs LDF of DF that were constructed in the lower level. In order to construct these DF, we define an affinity between folders by defining the pairwise diffusion distances between the lower level DF using the lower level localized affinity matrix. This diffusion distance is computed locally by raising the local sub-matrix of both DF by the power of the current level that was constructed in the hierarchy (advance in time). Fig. 3.4(a) shows the construction of the localized affinity matrix between pairs of DF. Fig. 3.4(b) shows the different distance metrics that are used for the construction of the localized affinity matrix between pairs of DF.

Once the affinity between DF is defined, we continue with the construction process of the upper level of the hierarchy as described in the construction of the bottom level of the hierarchy. This way, we advance in time and more levels are built in the hierarchy.

Fig. 3.5 shows the final bottom-up hierarchy.

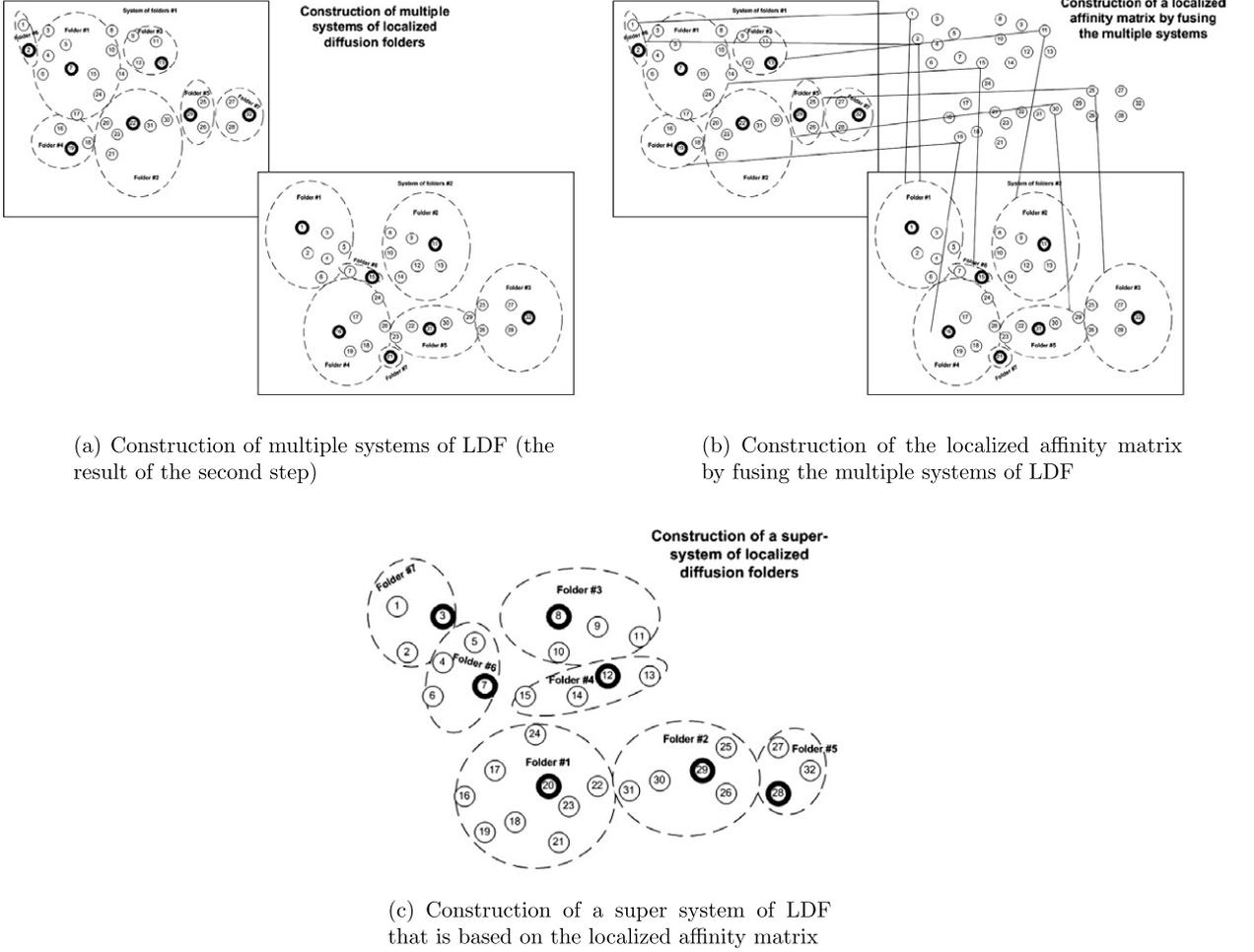


Fig. 3.3. Third step ($t = 1$): Construction of the localized affinity matrix and the corresponding LDF system.

3.2. Construction of the bottom level ($t = 1$) in the hierarchy: Detailed description

1. Input:

Let C be a matrix of size $m \times n$ of data points. m is the number of data points and n is the number of features in each data point. Each data point is a row vector in C .

2. Normalization of the matrix C :

All the features are transformed (normalized) into a common scale. A possible normalization option for the matrix C is by taking the logarithmic value of each feature. Assume r , $1 \leq r \leq m$, is a row in C denoted by $c_r \triangleq \{c_{ri}: 1 \leq i \leq n\}$. The normalized vector a_r is constructed by $a_r = \log(c_r)$, $r = 1, \dots, m$. A is the output matrix after normalizing each row $r = 1, \dots, m$ in the input matrix C .

3. Processing the normalized matrix A – construction of the similarity matrix \tilde{A} :

Denote the row vector i , $1 \leq i \leq m$, in the normalized matrix A by $a_i \triangleq \{a_{ik}: 1 \leq k \leq n\}$. The pairwise distances between data points in A are computed using a weight metric to produce \tilde{A} whose entries are \tilde{a}_{ij} . There are several optional distance metrics:

Euclidean distance metric: $\tilde{a}_{ij} \triangleq \left\{ \sqrt{(a_i - a_j) \cdot (a_i - a_j)^T} : i, j = 1, \dots, m \right\}$.

Weighted Euclidean distance metric: $\tilde{a}_{ij} \triangleq \left\{ \sqrt{\left(\frac{a_i - a_j}{w} \right) \cdot \left(\frac{a_i - a_j}{w} \right)^T} : i, j = 1, \dots, m \right\}$ where $w \triangleq \{w_k: 1 \leq k \leq n\}$ is a weight factor vector. As w_k becomes larger, the influence of the k -th feature on the distance between a_i and a_j becomes smaller.

Cosine distance metric: $\tilde{a}_{ij} \triangleq \left\{ \left(1 - \frac{a_i \cdot a_j^T}{\sqrt{a_i \cdot a_i^T} \cdot \sqrt{a_j \cdot a_j^T}} \right) : i, j = 1, \dots, m \right\}$.

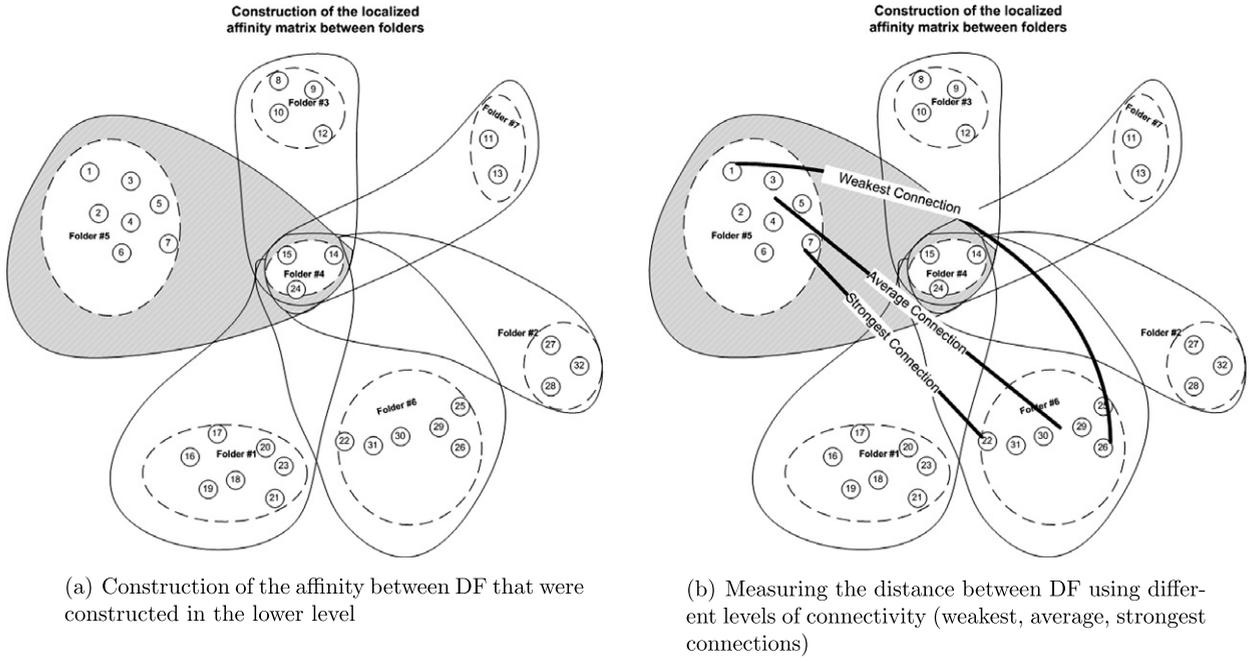


Fig. 3.4. Fourth step ($t > 1$): Construction of localized affinity matrix between DF.

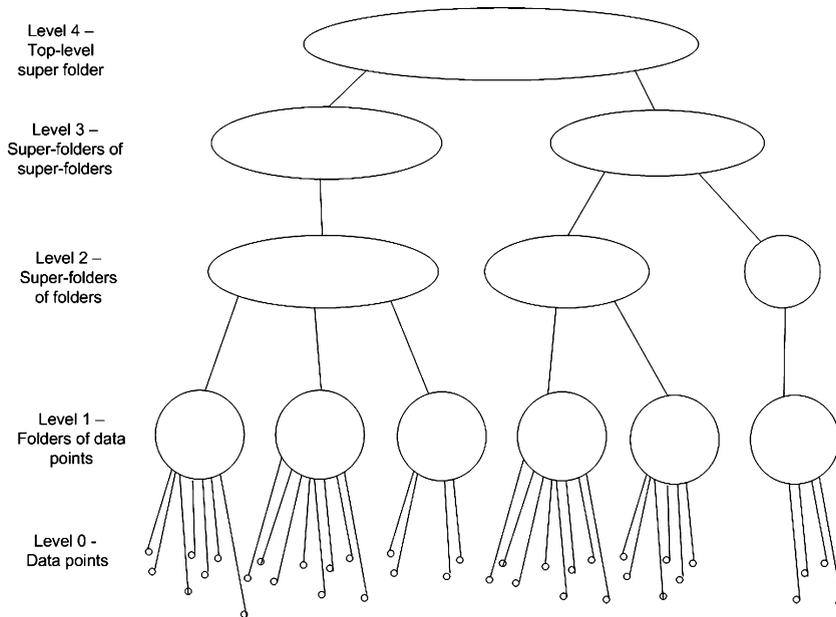


Fig. 3.5. Bottom-up hierarchy of the LDF systems ($t = 4$).

Mahalanobis distance metric: $\tilde{a}_{ij} \triangleq \sqrt{(a_i - a_j) \cdot \Sigma^{-1} \cdot (a_i - a_j)^T}$: $i, j = 1, \dots, m$ and Σ is the covariance matrix}.
 Σ can also be the features matrix.

As described in Section 1, W_ϵ is chosen to be $w_\epsilon(a_i, a_j) = e^{-\frac{\|a_i - a_j\|^2}{\epsilon}}$. Since an appropriate choice of ϵ is critical in the construction of the Gaussian kernel, we propose here a method that constructs an adaptive Gaussian kernel with an appropriate choice of ϵ . Let $A = \{a_1, \dots, a_m\}$ be a set of points in \mathbb{R}^n . Let $w_\epsilon(a_i, a_j) = e^{-\frac{\|a_i - a_j\|^2}{\epsilon}}$ be the imposed Gaussian kernel. For each data point $a_i \in A$, this Gaussian kernel pushes away from a_i all the data points that are already far away from a_i . On the other hand, it pulls towards a_i all the data points that are already close to a_i . This push-pull process is controlled by ϵ . Since ϵ is fixed for all the entries in W , it produces a coarse scaling control.

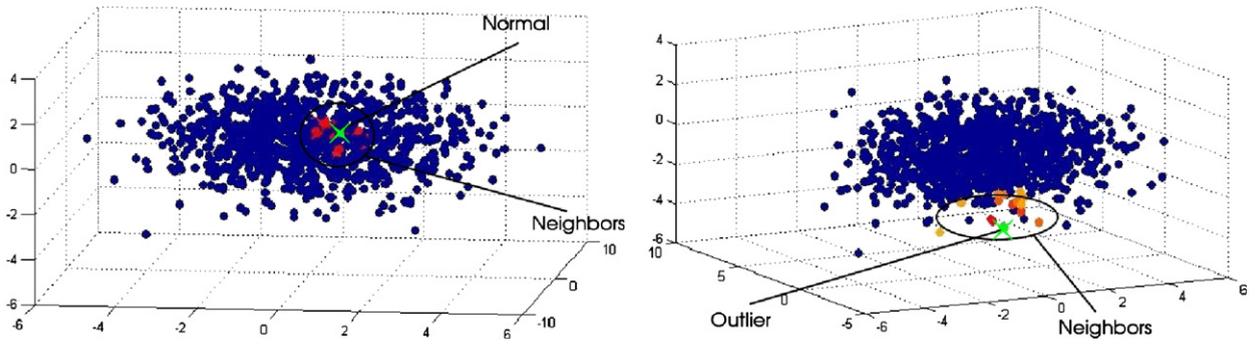


Fig. 3.6. Gaussian kernel with fixed ϵ . Left: Normal point and its neighbors. Right: Outlier point and its neighbors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

This scale control is obviously not optimal for all the entries in the Gaussian kernel since it does not reflect the local geometry of each data point in the graph. Fig. 3.6 shows an example of data points in \mathbb{R}^3 . In this example, $\epsilon = 0.02$ was chosen as the scale control. The orange/red points in this image are the neighbors of the green (cross) points according to the constructed Gaussian kernel (using the fixed scale control), where the scale control ϵ defines the neighborhood diameter. These are the points that were pulled towards the green points.

In the left image, we see that although the green point is in a very dense area, it has only few neighbors. In this case, we are interested in pulling more points towards the green point. This is achieved by selecting a larger ϵ (that provides a wider radius) since it will include more points in the neighborhood of the green point. However, in the right image, we see that although the green point is an outlier, it has relatively many neighbors where some of them are in areas that are well separated from the green point. In this case, we are interested in pushing more points away from the green point. This is achieved by selecting a smaller ϵ (that provides a narrower radius) since it will include less points in the neighborhood of the green point.

Therefore, a selection of an adaptive scale control is necessary in order to construct a more accurate Gaussian kernel that will express the local geometry of each data point in the graph.

In this section, we propose a method that constructs a two-phase adaptive Gaussian kernel by determining automatically an adaptive scale control for each point in the graph. This adaptive scale control is a weight function that has the following property: data points in dense areas will have a large weight (“bonus”) and data points in sparse areas will have a small weight (“penalty”). Thus, we define the weight function ω , which represents the local geometry around each data point, to be

$$\omega_{\epsilon_i}(a_i) = \int_{\check{A}_i} e^{-\frac{\|a_i - a_j\|^2}{\epsilon_i}} d\mu(a_j), \tag{3.1}$$

where \check{A}_i is a big cloud of points around a_i , μ is the distribution of the points in \check{A}_i , $a_j \in \check{A}_i$ and ϵ_i is an initial scale control that is adapted to \check{A}_i . In our experiments, we defined around each data point a_i a cloud \check{A}_i that included the $\frac{m}{3}$ nearest neighbors of a_i , hence, each cloud of points contained third of the data points.

Typically, two alternatives are considered in order to determine the scale controls ϵ_i [19,20]. In the first alternative, the same scale control is defined for all ϵ_i (see for example, [21–25]). However, these methods depend on uniform distribution of the data that is inadequate in practice where the data distribution is non-uniform. In the second alternative, local estimation of the scale controls is performed. A common strategy to scale control estimate is to take into consideration the distribution variances of the data points in different regions (see for example, [26–28]). These methods offer a better adaptivity to the data than a fixed scale control.

Therefore, in order to determine the scale control ϵ_i in Eq. (3.1), we calculate the variance of the distances between the point a_i to all the points a_j that belong to the cloud of points around it by

$$\epsilon_i = \sum_{a_j \in \check{A}_i} \frac{\| \|a_i - a_j\|^2 - \bar{\bar{A}}_i \|^2}{|\check{A}_i|}, \tag{3.2}$$

where $\bar{\bar{A}}_i = \sum_{a_j \in \check{A}_i} \frac{\|a_i - a_j\|^2}{|\check{A}_i|}$.

Assume $\omega_\epsilon(a_i)$, $i = 1, \dots, m$, defines an adaptive weight for each data point. However, since we construct an affinity matrix between pairs of data points, we have to define a pairwise weight function. We determine this way not only an adaptive scale for each point in A , but we also determine an adaptive scale for each pair of points. We define the pairwise weight function Ω_ϵ to be

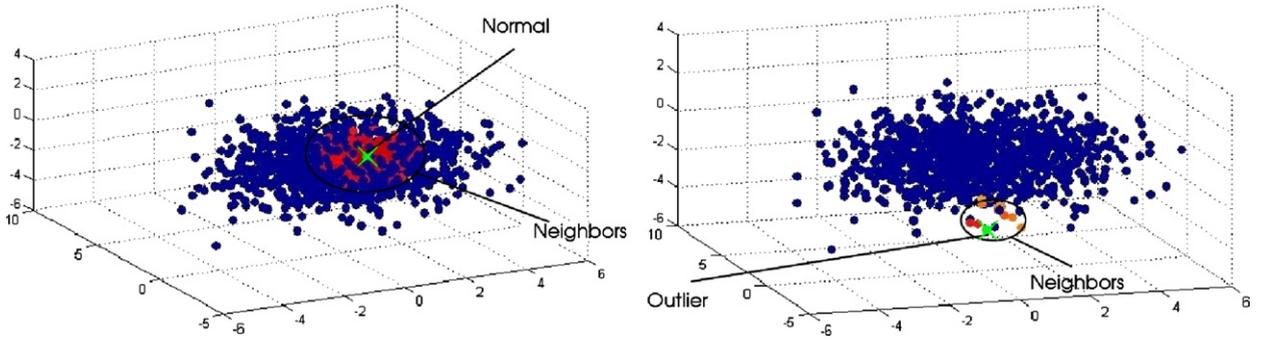


Fig. 3.7. Gaussian kernel with an adaptive scale control (Eq. (3.4)). Left: Normal point and its neighbors. Right: Outlier point and its neighbors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\Omega_\epsilon(a_i, a_j) = \sqrt{\int_A e^{-\frac{\|a_i - a_k\|^2}{\epsilon_i}} d\mu(a_k) \cdot \int_A e^{-\frac{\|a_j - a_k\|^2}{\epsilon_j}} d\mu(a_k)} = \sqrt{\omega_{\epsilon_i}(a_i) \cdot \omega_{\epsilon_j}(a_j)}. \tag{3.3}$$

Ω_ϵ satisfies the “bonus” and the “penalty” properties and it takes into consideration the distribution of the different regions in the data. Moreover, Ω_ϵ is symmetric and non-negative. Now, the adaptive Gaussian kernel W becomes

$$w_\epsilon(a_i, a_j) = e^{-\frac{\|a_i - a_j\|^2}{\sqrt{\omega_{\epsilon_i}(a_i) \cdot \omega_{\epsilon_j}(a_j)}}} = e^{-\frac{\|a_i - a_j\|^2}{\Omega_\epsilon(a_i, a_j)}}, \quad i, j = 1, \dots, m. \tag{3.4}$$

Since both the Euclidean distance metric and Ω_ϵ are symmetric and non-negative, the constructed kernel W is symmetric and non-negative as well. This adaptive scale control provides better and compact description of the local geometric properties of the pairwise distances matrix for X .

Fig. 3.7 shows an example of the same data points in \mathbb{R}^3 . In this example, we constructed the adaptive Gaussian kernel that was described by Eq. (3.4). In the left image, we see that the green point has more neighbors than when the fixed scale control is used (Fig. 3.6). The reason is that this green point is located in a very dense area and therefore it is close to many points. Hence, the adaptive scale control is much bigger than the initial scale control (a “bonus” was awarded) and as a result more points were pulled towards the green point. However, in the right image, we see that the green point has fewer neighbors than when the fixed scale control is used. The reason is that this green point is an outlier and therefore it is far from most of the data points. Therefore, the adaptive scale control is much smaller than the initial scale control (a “penalty” was assigned) and as a result more points were pushed away from the green point. These selections of the adaptive scale control provide an adaptive Gaussian kernel.

Fig. 3.8 shows the DM coordinates of the data points that use a Gaussian kernel with a fixed scale control and the proposed kernel with an adaptive scale control. As we can see, when we use a fixed scale control (the left image) we get a jelly-fish shape where the separation between normal (the body) and outliers (the tails) data points is unclear. However, when we use an adaptive scale control (the right image), we get a bell shape where the separation between the normal data points (the bell) and the outliers are clearer. This structure represents more accurately the geometry of the original data points.

Fig. 3.9 shows the performance of a Gaussian kernel with a fixed scale control and the proposed kernel with an adaptive scale control. For each set of data points in \mathbb{R}^2 (a single row in Fig. 3.9), we applied the DM with a different fixed scale control and with an adaptive scale control. We used the first three diffusion coordinates in order to cluster the data points using the k -means algorithm. In a single run of our implementation of k -means, the clustering was repeated several times. In each iteration, a new set of initial cluster centroid positions (seeds) were chosen. This implementation of the k -means algorithm provides a solution with the lowest value for the within-cluster sums of point-to-centroid distances. Therefore, this implementation takes into account the need to initialize with different seeds. The best chosen solution is based on a general criteria and it is not based on the target function. The left column of images in Fig. 3.9 shows the clustering results with an adaptive scale control. The remaining columns show the clustering results with different fixed scale control. We notice that the clustering performance with an adaptive scale control is very accurate. All the datasets were clustered correctly according to their natural clusters. However, in order to cluster correctly using a fixed scale control, it is necessary to choose manually an appropriate scale control for each dataset. In addition, a scale control, which is appropriate for one dataset (for example, 0.01 for the fourth dataset), is inappropriate for other datasets (for example, the second dataset requires a 100 times bigger scale control and the third dataset requires a 10 times bigger scale control). Moreover, for some datasets, all the fixed scale controls were inappropriate (for example, the fifth and the sixth datasets). Therefore, the right selection of a scale control is crucial in order to catch correctly and accurately the geometry of the data. The proposed kernel with an adaptive scale control does it correctly.

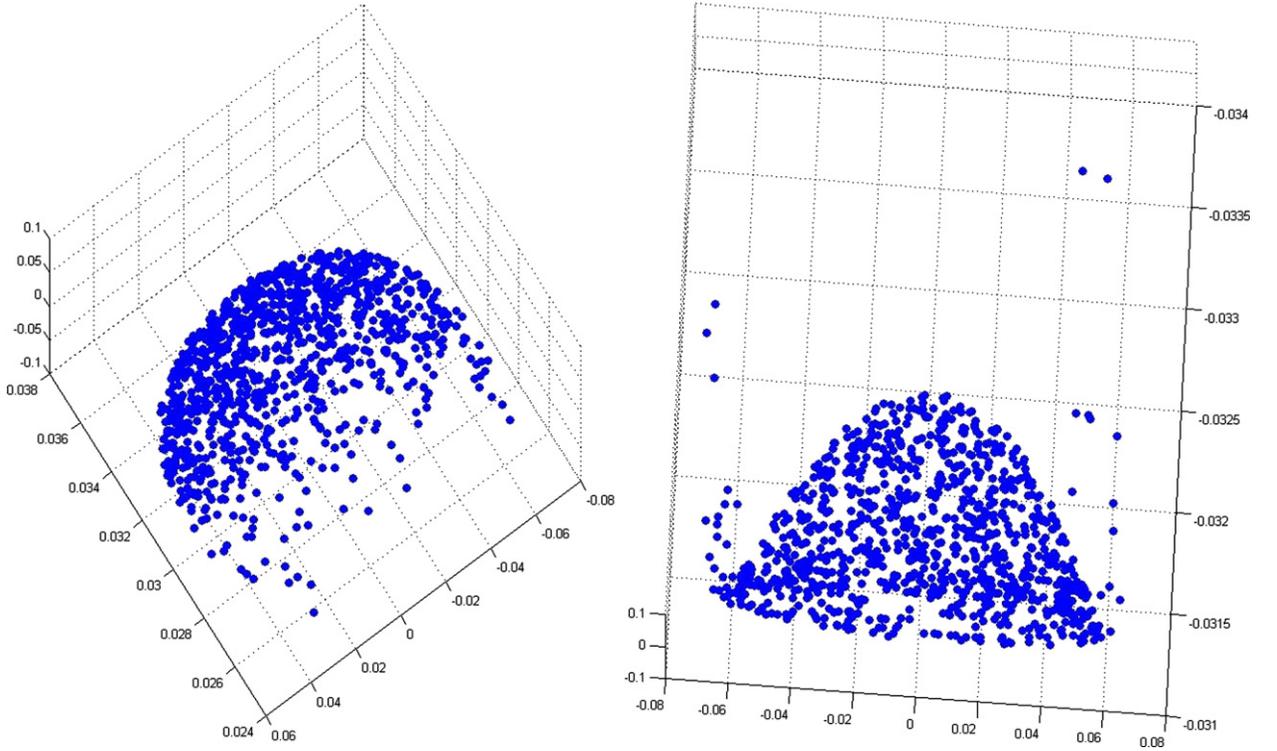


Fig. 3.8. Left: Application of the DM with fixed scale control. Right: Adaptive scale control (“bonus” and “penalty” are used in Eq. (3.4)).

4. Normalizing the Gaussian kernel W to become a Markov transition matrix:

The non-negativity property of W allows to normalize it into a Markov transition matrix $P^{(t)}$ where the states of the corresponding Markov process are the data points. This enables to analyze C as a random walk. W_{ij} is normalized into a matrix P_{ij}^t by one of the following methods:

Graph Laplacian matrix: $P_{ij}^{(t)} = \frac{W_{ij}}{\sqrt{\sum_{q=1}^m W_{iq}} \sqrt{\sum_{q=1}^m W_{jq}}}$.

Laplace–Beltrami matrix: First, we compute the graph Laplacian matrix $\tilde{P}_{ij}^{(t)} = \frac{W_{ij}}{\sqrt{\sum_{q=1}^m W_{iq}} \sqrt{\sum_{q=1}^m W_{jq}}}$. This process is repeated to get the Laplace–Beltrami matrix $P_{ij}^{(t)} = \frac{\tilde{P}_{ij}^{(t)}}{\sqrt{\sum_{q=1}^m \tilde{P}_{iq}^{(t)}} \sqrt{\sum_{q=1}^m \tilde{P}_{jq}^{(t)}}}$.

$P^{(t)}$ is a Markov matrix since the sum of each row in $P^{(t)}$ is 1 and $P_{ij}^{(t)} \geq 0$. Thus, $P_{ij}^{(t)}$ can be viewed as the probability to move from a_i to a_j in t time steps. $P^{(1)}$ is the initial affinity matrix of the graph.

5. Construction of the LDF $D^{(t)}$ using the affinity matrix $P^{(t)}$:

An initial partitioning of the data points into non-overlapping DF takes place. Let a_i be a random point in the dataset and let $N_\epsilon(a_i) \triangleq \{a_j : P_{ij}^{(t)} > \epsilon, 1 \leq j \leq m, i \neq j\}$ be the neighborhood of a_i . This neighborhood contains all the neighbors of the randomly selected point a_i where their distances to a_i , according to the affinity matrix $P^{(t)}$, is greater than ϵ . The DF is denoted by $D_i^{(t)} \triangleq \{a_j : a_j \in N_\epsilon(a_i), a_j \notin D_k^{(t)}, 1 \leq k \leq q, k \neq i, q \ll m\}$. Since the neighborhood $N_\epsilon(a_i)$ can contain points that were already associated with neighborhoods of other data points, we add to the DF of a_i only the data points that have not already been associated to any other DF. This way, the points are partitioned into non-overlapping DF. Next, we choose another non-associated random point a_l in the dataset where $a_l \notin D_k^{(t)}, 1 \leq k \leq q, q \ll m$ and the above is a repeated process in order to construct its DF $D_l^{(t)}$. The whole process is repeated until all the data points are associated with the constructed DF.

Once all the data points are assigned and we have a set $D_k^t, 1 \leq k \leq q, q \ll m$, of DF, we have to verify that each data point is associated with its nearest neighborhood in order to get accurate DF. This is achieved by reassignment of data points to their nearest neighborhood followed by DF reconstruction that takes place accordingly. For each data point $a_i, i = 1, \dots, m$, we calculate its average affinity $\mu(a_i, D_k^{(t)})$ to each DF $D_k^{(t)}, 1 \leq k \leq q, q \ll m$, according to the affinity matrix $P^{(t)}$, to be $\mu(a_i, D_k^{(t)}) = \frac{1}{|D_k^{(t)}|} \sum_{l=1}^{|D_k^{(t)}|} P_{iD_k^{(t)}}^{(t)}$, where $P_{iD_k^{(t)}}$ is the affinity between a_i to the l th data point in $D_k^{(t)}$.

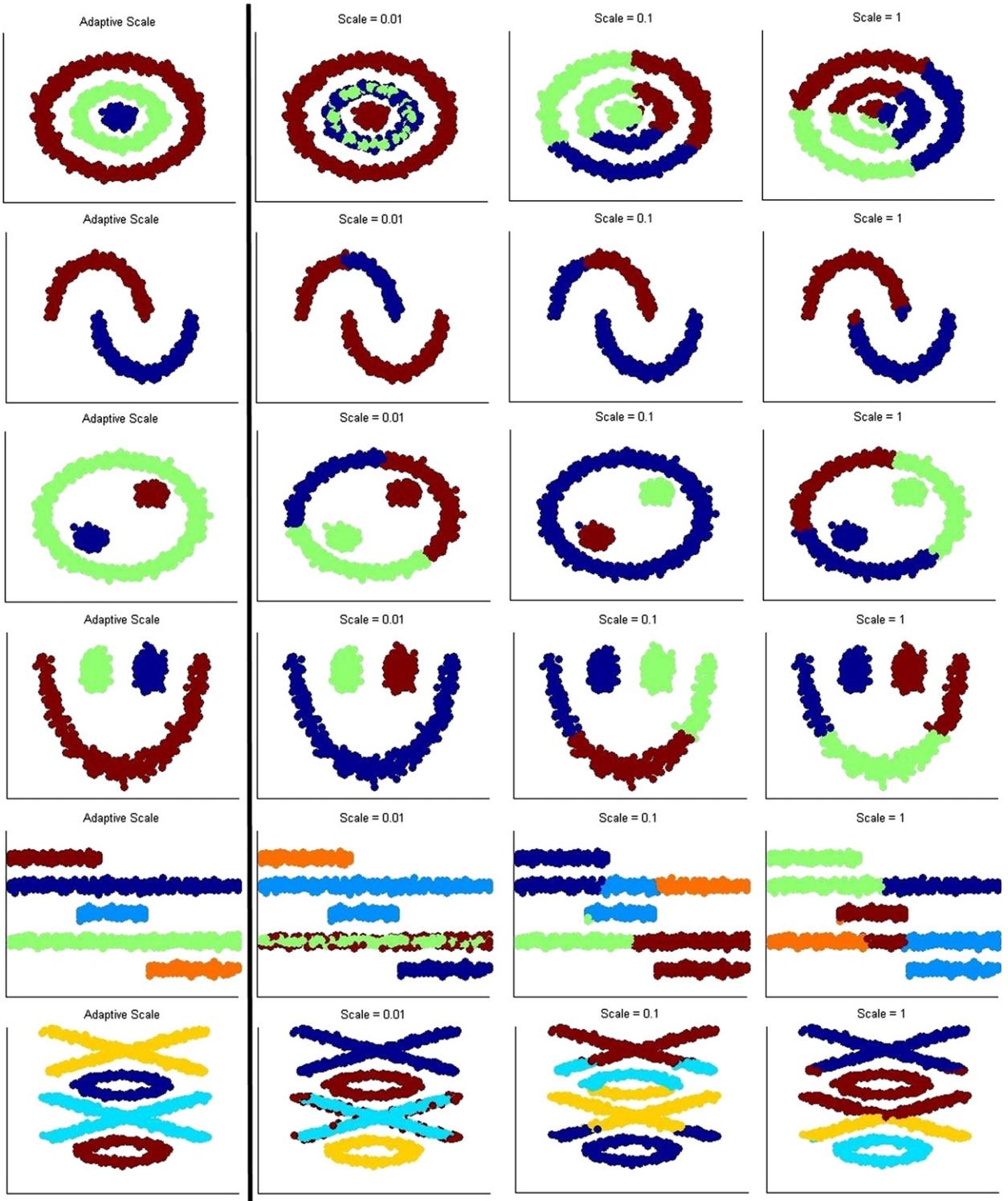
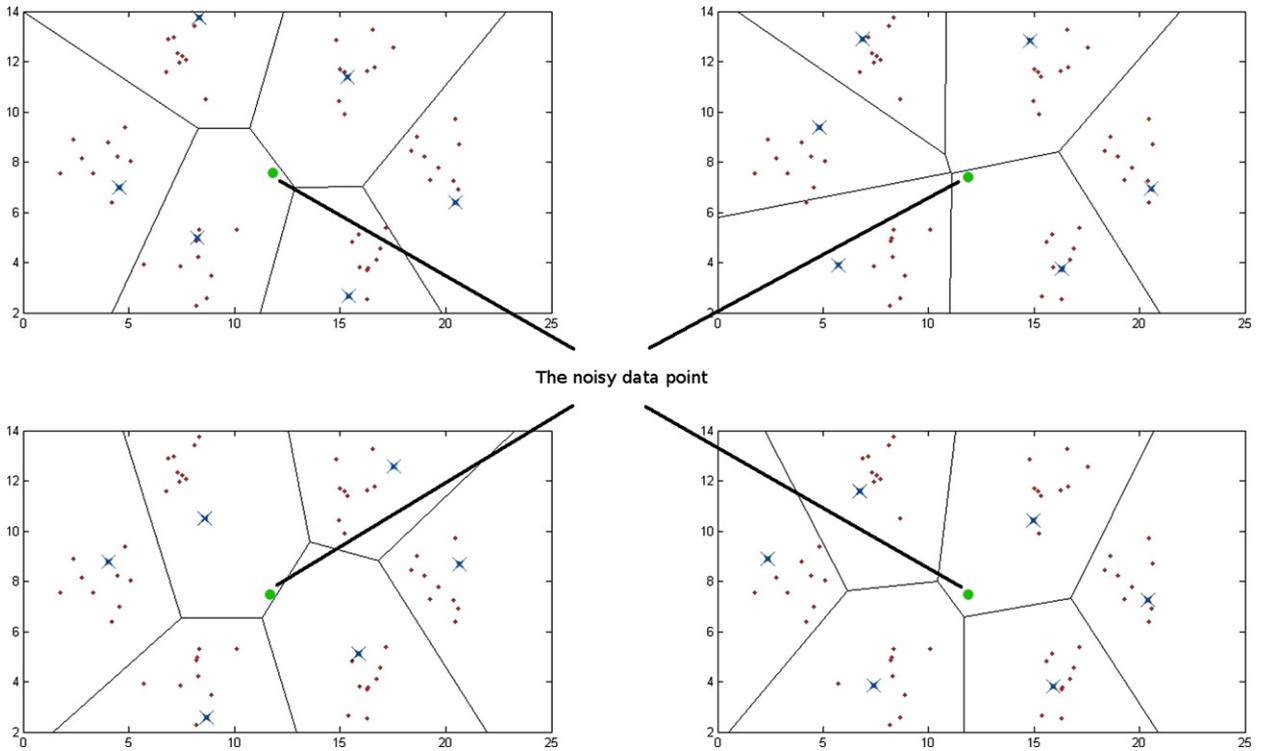


Fig. 3.9. Left column: Adaptive scale control (“bonus” and “penalty” are used). The three right columns: DM with different fixed scale control.

Finally, a_i is reassigned to a DF with the maximum average affinity μ . Once this process is applied to all data points, each data point is reassigned to one of the DF $D_k^{(t)}$, $1 \leq k \leq q$, $q \ll m$. Hence, at the end of this process, the members in each DF can be changed according to the reassignments. The whole reassignment process of data points and the folders reconstruction is repeated several times until the convergence of the diffusion folders is reached and all the folders become stable. In other words, either there are no more movements of data points between folders or data points



The noisy data point

Fig. 3.10. Four different Voronoi diagrams of the same dataset. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

movements are negligible. This iterative algorithm maximizes the sum of point-to-folder affinities, summed over all k folders. Each iteration consists of one pass through all points. The algorithm will converge to a local maximum, although there may be other local maxima with higher total sum of distances. The problem of finding the global maximum is solved in general in step 6, where multiple LDF systems are constructed. The result of this process is a Voronoi diagram [29] $D^{(t)}$ of LDF.

6. Construction of multiple systems $\tilde{D}^{(t)}$ of LDF:

In Step 5, we constructed a Voronoi diagram $D^{(t)}$ of LDF. Since this system was affected by the initial random selection of the points, we repeat this step r times until we get r different Voronoi systems of LDF. We denote the set of multiple systems of LDF by $\tilde{D}^{(t)} \triangleq \{D^{(t,k)} : k = 1, \dots, r\}$, where $D^{(t,k)}$ is the k th system of LDF. $\tilde{D}^{(t)}$ contains r systems of DF where the construction of each system of folders was initiated by different selections of random data points. In our experiments, we used $r = 10$ as the initial number of Voronoi systems of LDF.

7. Construction of the localized affinity matrix $\hat{P}^{(t)}$ – the “shake n bake” process:

We fuse the multiple Voronoi systems $\tilde{D}^{(t)}$ of LDF to get rid of the noise from the original affinity (“shake”). Then, a cleaner and more accurate affinity matrix $\hat{P}^{(t)}$ is constructed (“bake”). The key idea of this process is that two points are close to each other iff they are associated with the same DF in different systems (at least in the majority of the systems).

Fig. 3.10 shows four different Voronoi diagrams that were constructed according to different centers selections of the Voronoi cells. We can see that most of the time, although the centers (blue cross points) are changed, the data points (small red points) are associated with the same Voronoi cells. However, the noisy data point (large green circle) is associated with different Voronoi cells. This means that we need to construct and examine different Voronoi diagrams using different centers in order to find and eliminate non-consistent associations.

Therefore, we define the following metric space [30] (A, d) where $A = \{a_1, \dots, a_m\}$ is the set of points and d is a metric on A such that $d : A \times A \rightarrow \mathbb{R}$ is

$$d(a_i, a_j) = \begin{cases} 0 & a_i = a_j, \\ \frac{1}{2} & a_i \neq a_j \text{ and } a_j \in D_q^{(t)}(a_i), \\ 1 & a_i \neq a_j \text{ and } a_j \notin D_q^{(t)}(a_i), \end{cases}$$

where $D_q^{(t)}(a_i)$ is the DF q in the Voronoi system $D^{(t)}$ that contains a_i . The proof that (A, d) is a metric space is given in Appendix A.

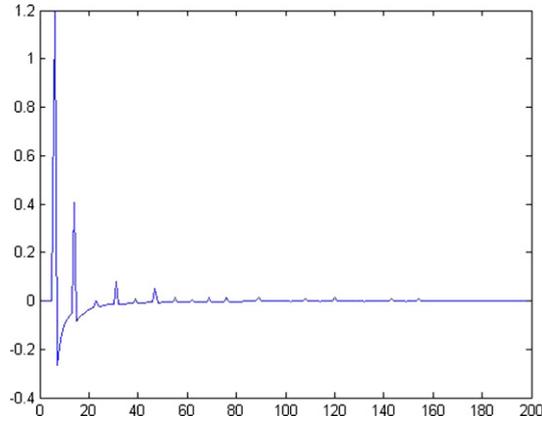


Fig. 3.11. The first derivative of Δ . The x is the number of Voronoi systems.

In order to fuse the multiple Voronoi systems of the DF $\tilde{D}^{(t)}$, we define the following metric space (A, d_μ) (the proof that (A, d_μ) is a metric space is given in Appendix B) where $A = \{a_1, \dots, a_m\}$ is the set of data points and d_μ is a metric on A such that $d_\mu : A \times A \rightarrow \mathbb{R}$, $d_\mu(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j)$, where $|\tilde{D}^{(t)}|$ is the number of Voronoi systems in $\tilde{D}^{(t)}$ and $d_k(a_i, a_j)$ is the distance between a_i to a_j according to the metric d and the Voronoi system $D^{(t,k)}$.

We define the localized affinity matrix \hat{P} as $\hat{P}_{ij}^{(t)} = 1 - d_\mu(a_i, a_j)$. The affinity $\hat{P}^{(t)}$ is localized and it reduces the noise by eliminating rare connections between data points. This affinity imposes a new geometry on the data according to the actual different partitions of the data. Instead of defining a general affinity on the data, we let the data define the local affinity by itself. We normalize this affinity into a Markov transition matrix (normalized graph Laplacian or normalized Laplace–Beltrami) as was described in Section 3.2 Step 5.

8. Refinement of the localized affinity matrix $\hat{P}^{(t)}$:

In Step 7, we fused the r Voronoi systems \tilde{D}^t of LDF, which were constructed in Step 6 where $r = 10$ was proposed as the initial number of Voronoi systems of LDF. In this step, we refine the localized affinity matrix $\hat{P}^{(t)}$ by fusing more Voronoi systems of the LDF. For each new Voronoi system, which is constructed as was described in Step 5, we update the localized affinity matrix $\hat{P}^{(t)}$ by constructing it as described in Step 7 using the previous r Voronoi systems and the new Voronoi system. We denote by $\hat{P}_r^{(t)}$ the localized affinity matrix $\hat{P}^{(t)}$ that is constructed using r Voronoi systems. In each iteration, which refines the localized affinity matrix, we increase r by 1 and construct the corresponding $\hat{P}_{r+1}^{(t)}$. Then, we measure the Hilbert–Schmidt norm of the difference between $\hat{P}_r^{(t)}$ and $\hat{P}_{r+1}^{(t)}$ as follows:

$$\Delta(r, r+1) \triangleq \sqrt{\sum_{i=1}^m \lambda_i^2}, \text{ where } \lambda_i \text{ is the } i\text{th singular value of the difference matrix } (\hat{P}_r^{(t)} - \hat{P}_{r+1}^{(t)})^T (\hat{P}_r^{(t)} - \hat{P}_{r+1}^{(t)}).$$

When the function $\Delta(r, r+1)$, $r = 10, 11, \dots$ decays to zero, the refinement process is terminated and the final localized affinity matrix $\hat{P}^{(t)}$ is defined as $\hat{P}_r^{(t)}$ (the last constructed localized affinity matrix).

We now illustrate the refinement of the localized affinity matrix on the example in Section 1.1. We begin from Fig. 1.1. We used the original affinity to construct and to refine the localized affinity matrix, as described in previous steps. Fig. 3.11 shows the first derivative of the function Δ , $r = 1, \dots, 200$. We see that for $r = 55$, this function converges and the changes between consecutive affinities are minor. Therefore, 55 Voronoi systems are required to construct the localized affinity matrix $\hat{P}^{(t)}$.

Fig. 1.2 in Section 1.1 shows the LDF affinity using different number of Voronoi systems. When 10 systems are used, several original connections are lost. When the number of fused systems increases, more original connections are re-constructed and the affinity is stabilized after 60 Voronoi systems (as it is determined by the first derivative of Δ) are fused. In addition, the noisy line in the original affinity (Fig. 1.1) was eliminated from each of the LDF affinities. At the end of this process, each data point was connected to the first point on its right and left sides on the circle with an average normalized weight of 0.2 and connected to the second point on its right and left sides on the circle with an average normalized weight of 0.13. These are exactly the normalized weights that we get when repeating this process on a circle that does not contain noisy edges at all.

Fig. 1.3 in Section 1.1 shows how the original diffusion algorithm handles this affinity.

9. Construction of the super-system $\hat{S}^{(t)}$ of LDF using $\hat{P}^{(t)}$:

Once we have the localized affinity matrix $\hat{P}^{(t)}$, which was constructed from the set of multiple $\tilde{D}^{(t)}$ systems of LDF, we can construct the super-system $\hat{S}^{(t)}$ of LDF. The construction of $\hat{S}^{(t)}$ is similar to the $D^{(t)}$ construction of the LDF that was described in Section 3.2 Step 6 but instead of using the original affinity matrix $P^{(t)}$ we use the improved affinity matrix $\hat{P}^{(t)}$. $\hat{S}^{(t)}$ is a super-system of the systems $\tilde{D}^{(t)}$ of LDF. This last step finalizes the construction of the bottom

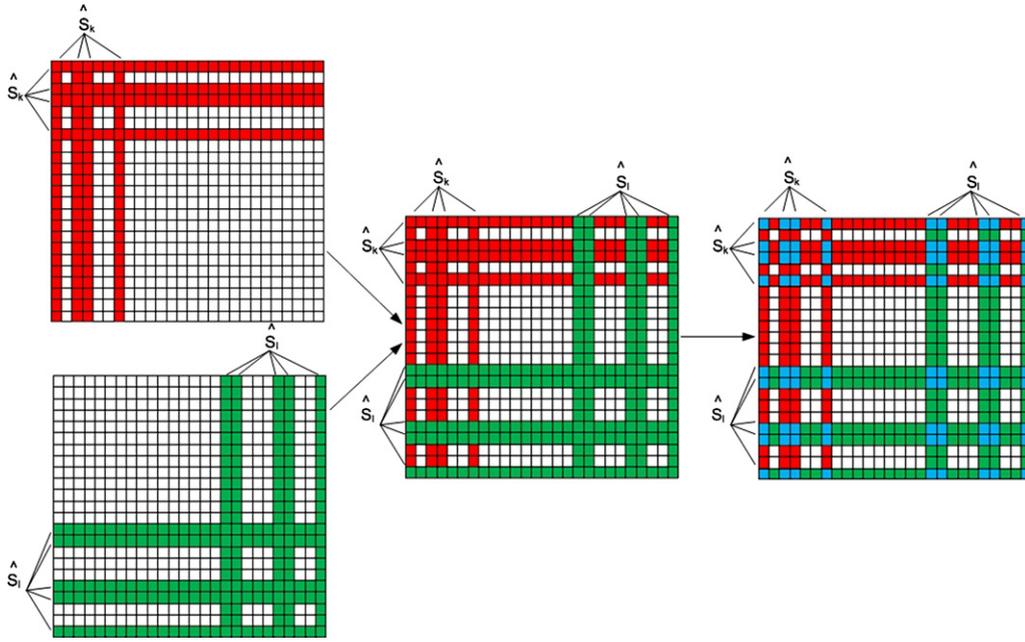


Fig. 3.12. Selection of the sub-matrix $\hat{P}_{kl}^{(t-1)}$ from the affinities between points in $\hat{S}_k^{(t-1)} \cup \hat{S}_l^{(t-1)}$.

level of the hierarchy of the LDF. An initial partitioning (clustering) $\hat{S}^{(t)}$ of the data into LDF is achieved. These LDF are the input to the next upper level construction of the LDF hierarchy. The final localized affinity matrix $\hat{P}^{(t)}$ is also an input to the next upper level.

3.3. Construction of the upper levels in the hierarchy

1. Input

Let $\hat{S}^{(t-1)}$ be the LDF and let $\hat{P}^{(t-1)}$ be the localized affinity matrix that were constructed in the lower level $t - 1$ in the LDF hierarchy.

2. Construction of the localized affinity matrix $P^{(t)}$ between the lower-level folders in $\hat{S}^{(t-1)}$

Let $|\hat{S}^{(t-1)}|$ be the number of LDF in $\hat{S}^{(t-1)}$. In order to construct a localized affinity matrix $P^{(t)}$ between the lower-level folders in $\hat{S}^{(t-1)}$, the pairwise diffusion distances between each pair of folders in $\hat{S}^{(t-1)}$ are computed. Let $\hat{S}_k^{(t-1)}$ and $\hat{S}_l^{(t-1)}$, $1 \leq k, l \leq |\hat{S}^{(t-1)}|$, be two different folders in $\hat{S}^{(t-1)}$. Then, the sub-matrix of $\hat{P}^{(t-1)}$, which contains only the affinities between all the data points in $\hat{S}_k^{(t-1)} \cup \hat{S}_l^{(t-1)}$ to all the data points in $\hat{S}_k^{(t-1)} \cup \hat{S}_l^{(t-1)}$, is denoted by $\hat{P}_{kl}^{(t-1)}$.

Fig. 3.12 demonstrates the selection of the sub-matrix $\hat{P}_{kl}^{(t-1)}$. This sub-matrix includes the blue entries in the right image.

This localized affinity sub-matrix is raised to a power of 2^t , denoted by $Q_{kl}^{(t)} \triangleq \hat{P}_{kl}^{(t-1)2^t}$, to diffuse the affinities in time. Note that we use the power of 2^t rather than the power of t to achieve a faster propagation and a faster diffusion of the affinities. This way, each level in the hierarchy is significantly different from the lower level.

Only the affinities between all the points in both folders propagate in time. This way, we eliminate distinct (rare) connections between them while increasing the accuracy of the distance metric. $Q_{kl}^{(t)}$ enables to preserve only local connections. Since we are interested in the diffusion connectivity between the two folders, only the sub-matrix of $Q_{kl}^{(t)}$, which contains the affinities between the points in $\hat{S}_k^{(t-1)}$ to the points in $\hat{S}_l^{(t-1)}$, is needed. This sub-matrix is denoted by $\hat{Q}_{kl}^{(t)}$.

Fig. 3.13 demonstrates the process of generating the sub-matrix $\hat{Q}_{kl}^{(t)}$. This sub-matrix includes the yellow entries in the right image.

Last, the affinity $P_{kl}^{(t)}$ between the folders $\hat{S}_k^{(t-1)}$ and $\hat{S}_l^{(t-1)}$ is defined by one of the following metrics:

Fastest random runner: $P_{kl}^{(t)} = \max_{1 \leq i \leq |\hat{S}_k^{(t-1)}|} \max_{1 \leq j \leq |\hat{S}_l^{(t-1)}|} \hat{Q}_{kl_{ij}}^{(t)}$. This metric expresses the fastest path between the two DF. Since this path is determined by the application of multiple random walks between one folder to another, the term *random runner* is used.

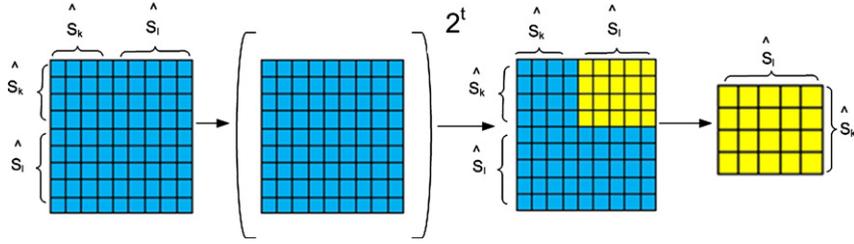


Fig. 3.13. Generating the sub-matrix $\hat{Q}_{kl}^{(t)}$.

Slowest random runner: $P_{kl}^{(t)} = \min_{1 \leq i \leq |\hat{S}_k^{(t-1)}|} \min_{1 \leq j \leq |\hat{S}_l^{(t-1)}|} \hat{Q}_{kl_{ij}}^{(t)}$. This metric expresses the slowest path between the two DF.

Average random runner: $P_{kl}^{(t)} = \frac{\sum_{i=1}^{|\hat{S}_k^{(t-1)}|} \sum_{j=1}^{|\hat{S}_l^{(t-1)}|} \hat{Q}_{kl_{ij}}^{(t)}}{|\hat{S}_k^{(t-1)}| |\hat{S}_l^{(t-1)}|}$. This metric expresses the average path between the two DF.

At the end of this process, $P^{(t)}$ is a non-negative and symmetric matrix that contains the pairwise distances between any pair of folders. The size of $P^{(t)}$ is $|\hat{S}^{(t-1)}| \times |\hat{S}^{(t-1)}|$. $P^{(t)}$ is normalized into a Markov transition matrix (normalized graph Laplacian or normalized Laplace–Beltrami) as described in Step 5 of Section 3.2. $P^{(t)}$ is the localized affinity matrix between the DF of the lower-level ($t - 1$) in the hierarchy.

3. Construction of the upper LDF level

The rest of the process, which constructs the upper level of the current level, is similar to the construction of the bottom level of the hierarchy that was described in Steps 6–9 of Section 3.2. However, the input data for this process is a set of DF of points $\hat{S}^{(t-1)}$ (instead of a set of points as in the bottom level) and the initial localized affinity for this process is the localized affinity matrix $P^{(t)}$ between DF in $\hat{S}^{(t-1)}$. Therefore, in this level, we cluster folders and not points. At the end of this construction of each upper level in the hierarchy, a higher-level partition $\hat{S}^{(t)}$ of the data into LDF is achieved. These LDF are the input for the next upper level ($t + 1$) in the LDF hierarchy. In addition, the final localized affinity matrix $\hat{P}^{(t)}$, which was constructed in this process, is the input affinity matrix for the next upper level ($t + 1$).

The process of constructing the upper levels in the hierarchy is repeated up to the root of the hierarchy.

4. Experimental results

The proposed LDF methodology for hierarchical clustering was tested on several datasets that belong to different domains:

- Network protocols dataset: clustering and classification of network packets (Section 4.1);
- Wine dataset: wine recognition and classification (Section 4.2);
- Iris dataset: clustering of iris plants (Section 4.3);
- Image processing: image denoising and restoration (Section 4.4).

4.1. The network protocols dataset

This is a proprietary non-public dataset. This dataset contains records of network activities from different applications such as Skype, Google Talk, ICQ, Windows Messenger, Microsoft Outlook, Mozilla Thunderbird, Limewire, Gnuceus, eMule, eDonkey2000, BitLord, uTorrent, etc. Each record contains the numerical statistics of a single network activity (for example, a single chat) from a single application. Each record contains the statistics of 30 parameters. These features include the duration of the activity, the number of bytes the client sent to the server and received from the server, the upload and download bit rate, the number of data packets and control packets, etc. The dataset, which we used in the experimental evaluation, contains 5500 records where each record belongs to one of 16 applications. Our goal was to hierarchically cluster the records according to their application, protocols and meta-protocols. In the lower levels of the hierarchy, the records expected to be clustered according to their applications and meta-applications (for example, Windows Messenger and Windows Live Messenger). In the upper levels of the hierarchy, the records should be clustered according to their protocols (for example, chat, Voip, etc.) and their meta-protocols (for example, symmetric and asymmetric communication). Fig. 4.1 presents a diagram of a possible hierarchy from such dataset.

In order to evaluate the quality of the clustering produced by our proposed method, we compared its performance with the performance of several known algorithms and methods. We measured the quality and the accuracy of the clustering algorithms as follows: Let $X = \{x_1, \dots, x_m\}$ be a set of n -dimensional points in \mathbb{R}^n where each point $x_i \in X$ is described by n variables $x_i = \{x_i^1, \dots, x_i^n\}$. Let $L = \{l_1, \dots, l_q\}$ be a set of different classes. For each n -dimensional point $x_i \in X$, the

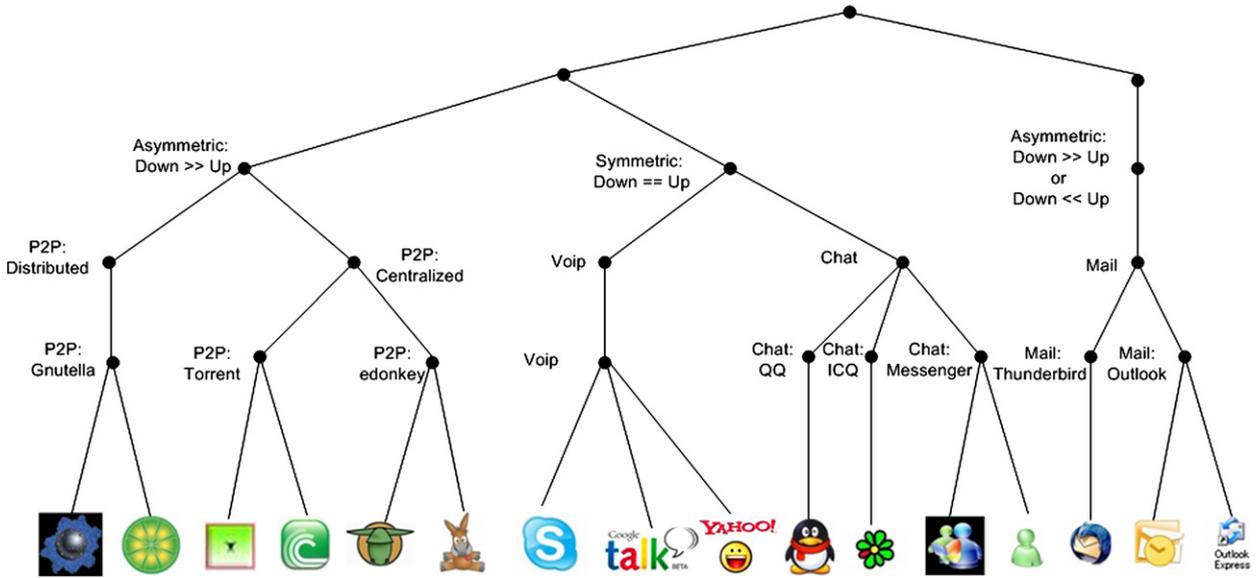


Fig. 4.1. A possible hierarchy of network applications.

corresponding label $y_i, y_i = l_j, 1 \leq j \leq q$, is assigned. Therefore, $Y = \{y_1, \dots, y_m\}$ is a set of labels for the dataset X . Y is used only for measuring the quality of the clustering algorithms and not for the clustering process itself.

Let f be a clustering algorithm to be evaluated. Let k be the number of clusters that f generates. Then, f_k is a clustering algorithm that associates each $x_i \in X, i = 1, \dots, m$, with one of the clusters $c_r \in C, r = 1, \dots, k$, where k is the number of clusters in C . Each $c_r \in C$ is labeled according to the majority of the records in the cluster. Formally, let $B_{c_r}^i \triangleq \sum_{x_p \in c_r} \delta(x_p, l_i), 1 \leq i \leq q, 1 \leq r \leq k$, where

$$\delta(x_p, l_i) \triangleq \begin{cases} 1 & \text{if } y_p = l_i, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the label of each cluster $c_r, r = 1, \dots, k$, is denoted by $M_{c_r} \triangleq \{l_j: \max_{1 \leq i \leq q} B_{c_r}^i\}$.

In order to evaluate the quality of the clustering algorithms, we measure the number of records in each cluster whose labels is equal to the label of the majority of the records in the cluster where the majority of the records is at least $P\%$ from the total number of records in the cluster. This measure determines the purity of the cluster (the intra-cluster accuracy). This P -purity accuracy is defined by

$$OP_k^P \triangleq \frac{\sum_{r=1}^k M_{c_r}}{k} = \frac{\sum_{r=1}^k \max_{1 \leq i \leq q} B_{c_r}^{iP}}{k}, \tag{4.1}$$

where k is the total number of clusters and $B_{c_r}^{iP}$ is defined as:

$$B_{c_r}^{iP} \triangleq \begin{cases} 1 & \text{if } \frac{B_{c_r}^i \cdot 100}{|c_r|} > P, \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

In Figs. 4.2–4.4, the X -axis represents different P -purity levels and the Y -axis represents the percentages of clusters that gained these levels.

First, we analyze the effect of t on the quality of the clustering. Fig. 4.2 shows the comparison between P -purity results from the LDF algorithm for $t = 1, 2, 3$ (first, second and third time steps which is also first, second and third levels in the hierarchy of the DF, respectively). In this experiment, we had only 3 levels in the hierarchy.

As t increases, the accuracy improves accordingly. The reason is that at the bottom of the hierarchy ($t = 1$), we still have some clustering errors and as a result the accuracy of some DF is relatively low. However, as t increases and DF are merged by the LDF methodology, then these clustering errors become negligible and the overall accuracy improves.

Since our method aims to improve the clustering via the application of the DM methodology, we compare between the results from the LDF and from the DM algorithms. We first applied DM to the data and then we clustered the points in the embedded space. We used diffusion time $t = 1$ for the Diffusion Maps since it achieved the best results. To evaluate the performance of the DM, the kernels were constructed according to different distance metrics such as Euclidean (L_2), Mahalanobis, Cosine and Weighted Euclidean (WL_2). Fig. 4.3 shows this comparison between the P -purity results from the application of the LDF methodology and from the application of the DM when different distance metrics were used.

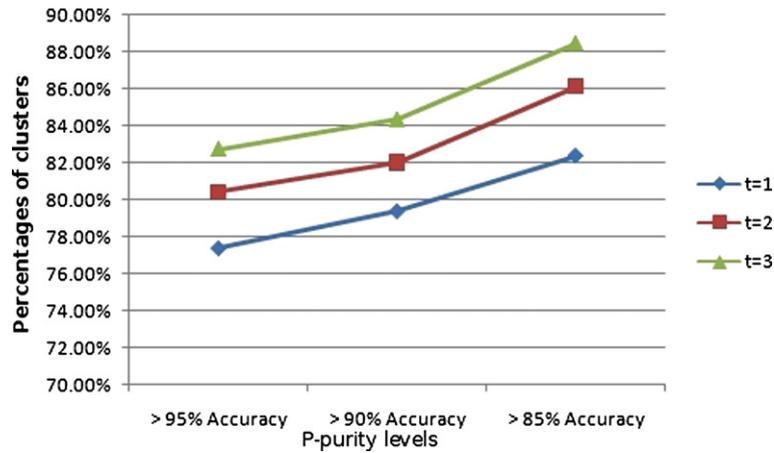


Fig. 4.2. Comparison between the intra-cluster accuracy results for different t values.

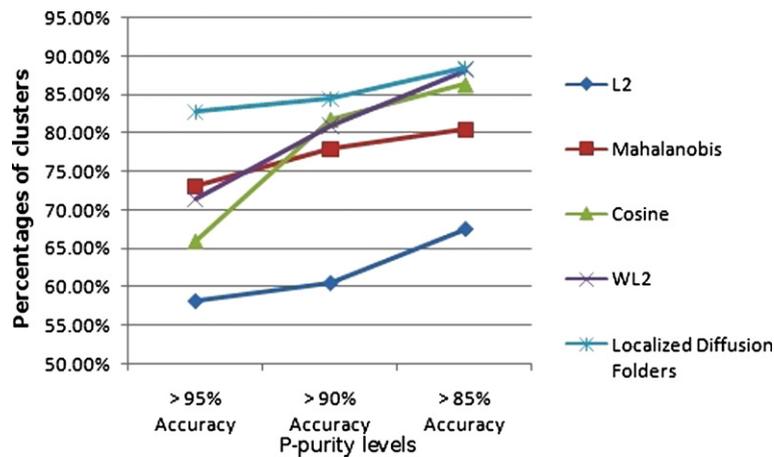


Fig. 4.3. Comparison between intra-cluster accuracy results from the application of DM and LDF ($t=3$) algorithms.

We see that for 95%-purity, the LDF algorithm produces better accurate results by about 15% than the best DM results. For 90%-purity, the LDF algorithm produces better accurate results by about 5% and for 85%-purity we get about the same results. This means that the LDF algorithm generates more accurate and purer clusters. Note that WL_2 achieved better results than L_2 since some parameters are more important than others and thus contribute more while some parameters mask the others. Hence, a proper weights assignment to the parameters improves the accuracy of WL_2 in comparison to L_2 .

Last, we compared between the accuracy that the LDF algorithm produces with the accuracy that k -means, BIRCH, and CURE clustering algorithms produce. Each of the compared algorithms was applied 30 times (each time using different initial points) and the average performances were used as the accuracy of each algorithm. Fig. 4.4 shows the comparison between the P -purity results from the LDF algorithm and from the different clustering algorithms.

We can see that the LDF algorithm outperforms the other clustering algorithms while achieving the best results for different P -purity levels.

4.2. The wine dataset [31]

This is a result from a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three wines types such as Alcohol, Malic acid, Magnesium, Hue and Proline. The dataset contains 178 wines that belong to three classes (the three types of wines): 59 wines belong to class 1, 71 wines belong to class 2 and 48 wines belong to class 3. In this experiment, the affinity matrix between wines was constructed according to the Euclidean distance of the log value of each wine. Then, the hierarchical LDF is constructed accordingly. In the bottom level of the hierarchy ($t=1$) we had 22 DF. In the second level ($t=2$) we had 10 DF. In the third level ($t=3$) we had 7 DF and in the fourth level ($t=4$) we had 5 DF.

The overall accuracy in each level was measured as follows: each DF was labeled according to the majority of the points that have the same label. The overall accuracy is the ratio between the total number of points that have the same label as the majority (in their DF) and the total number of points in the dataset.

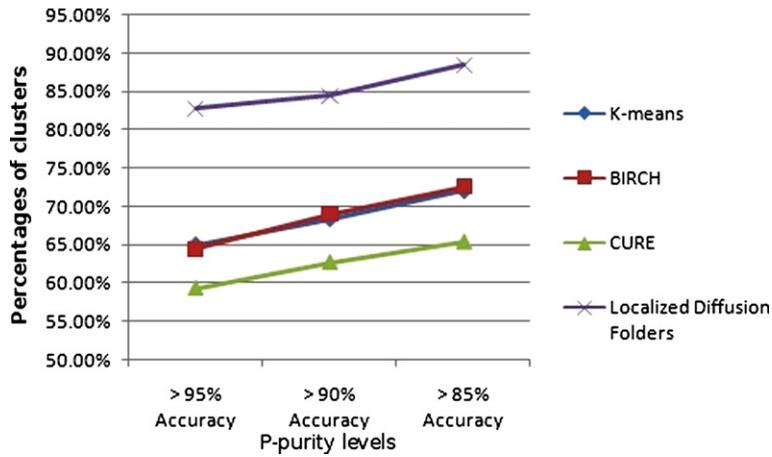


Fig. 4.4. Comparison between the intra-cluster accuracy results from *k*-means, BIRCH, CURE and LDF ($t = 3$) algorithms.

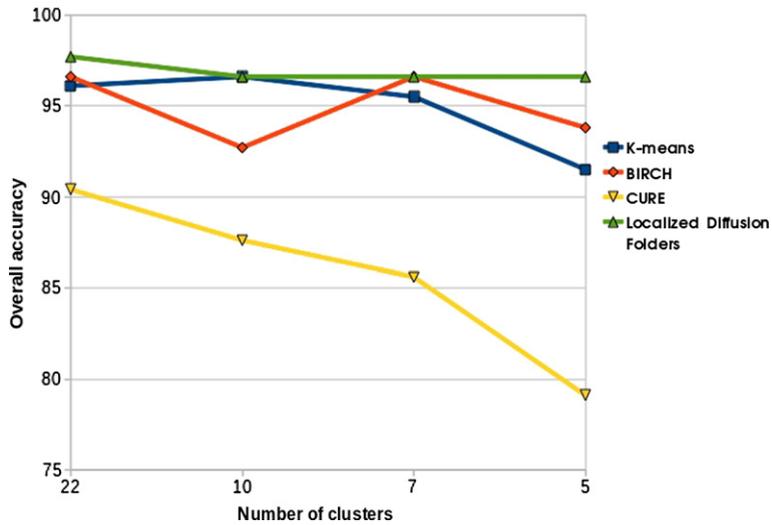


Fig. 4.5. Comparison between the overall accuracy results from *k*-means, CURE, BIRCH and LDF algorithms.

We compared the overall accuracies of the LDF algorithm with the accuracy of *k*-means, CURE and BIRCH clustering algorithms. The average overall accuracy, determined by over 30 runs of each algorithm, was evaluated for different number of clusters (22, 10, 7 and 5). Fig. 4.5 shows the comparison results. The X-axis in this figure represents different number of clusters and the Y-axis represents the overall accuracy.

We see that for 22 clusters ($t = 1$ in the LDF algorithm), the LDF algorithm is more accurate than BIRCH and *k*-means by 1.1% and by 1.8%, respectively. For 10 clusters ($t = 2$ in the LDF algorithm), the LDF algorithm is more accurate than BIRCH and as accurate as *k*-means. For 7 clusters ($t = 3$ in the LDF algorithm), the LDF algorithm is more accurate than *k*-means and as accurate as BIRCH. For 5 clusters ($t = 4$ in the LDF algorithm), the LDF algorithm is more accurate than BIRCH and *k*-means by 3.1% and by 5.7%, respectively. Note that CURE achieved the worst results. The overall accuracy of the LDF algorithm for this dataset was better than the compared algorithms.

4.3. The iris dataset [32]

This is perhaps the best known dataset in the pattern recognition literature. It contains information about three types of iris plants. The plants are described by four variables (sepal, petal length and width). This dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant (Setosa, Versicolour, and Virginica). One class is linearly separable from the other two. The latter are not linearly separable from each other.

We added Gaussian noise, where the signal-to-noise ratio per sample is -4 dB, to the original Iris dataset in order to determine its effect on different clustering algorithms. We measured the overall accuracy as described in Section 4.2, where each algorithm clustered the data into 3 groups. We compared between the accuracies of the LDF, *k*-means, CURE and BIRCH clustering algorithms. Table 4.1 shows these comparisons. Each clustering algorithm was applied 30 times using different

Table 4.1Comparison between the overall accuracies from k -means, CURE, BIRCH and LDF algorithms.

Algorithm	Worst accuracy	Best accuracy
LDF	84.2105	90.1316
BIRCH	66.2252	89.35
CURE	60.7368	79.7368
k -means	65.7895	89.4040

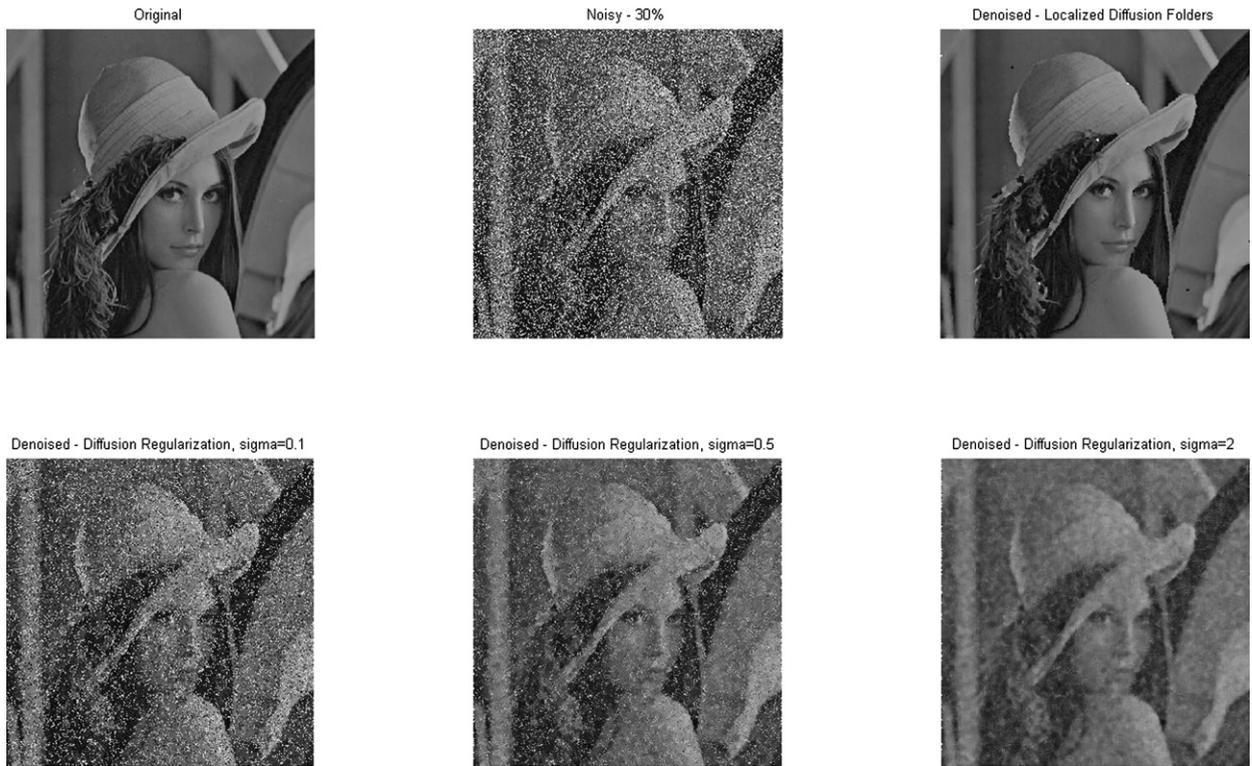


Fig. 4.6. Performance comparison between the denoising results from the application of the LDF algorithm and the diffusion regularization algorithms. The top-left image is the original image. The top-middle image is the noised image. We used a salt & pepper noise where 30% of the original pixels were replaced by salt (white) or pepper (black). The top-right image is the result from the application of the LDF based denoising algorithm (as was described above). The bottom images are the results from the application of the diffusion regularization algorithm using different σ (scale control) values while constructing the Gaussian kernel (as was described in [33]).

set of initial points and the overall accuracy was measured each time. In order to show the sensitivity of each algorithm to the added noise, Table 4.1 presents the best result (denoted best accuracy) and the worst result (denoted worst accuracy) as obtained by each algorithm.

For the worst case, we see that the LDF algorithm is more accurate than BIRCH, CURE and k -means by 21.35%, 27.87% and 21.87%, respectively. In this case, BIRCH, CURE and k -means algorithms failed to cluster the noisy dataset. For the best case, we see that the LDF algorithm is more accurate than BIRCH, CURE and k -means by 0.87%, 11.53% and 0.81%, respectively. For this noisy dataset, the overall accuracy of the LDF algorithm was better than the compared algorithms.

4.4. Image denoising and restoration

We used the LDF algorithm for denoising and restoring images as follows: first, we represent each pixel in the image by a window of 5×5 neighbors around it. This way, each pixel is transformed into a 25-dimensional vector (mega-pixel). Then, we moved with a sliding window of 9×9 mega-pixels over the image in order to determine the value of the center pixel of each sliding window. Each mega-pixel in 9×9 window corresponds to 25-dimensional vector that represents this pixel. The following process is applied to each sliding window: In the beginning we constructed the hierarchical LDF of its 9×9 mega-pixels. This way, the 81 mega-pixels are clustered into a hierarchy of folders. Then, the original value of the central pixel (in the 9×9 window) is replaced with the average value of the pixels from the largest folder in the third level of the hierarchy. The average value is determined by the original values of the pixels that correspond to the mega-pixels in this largest folder.

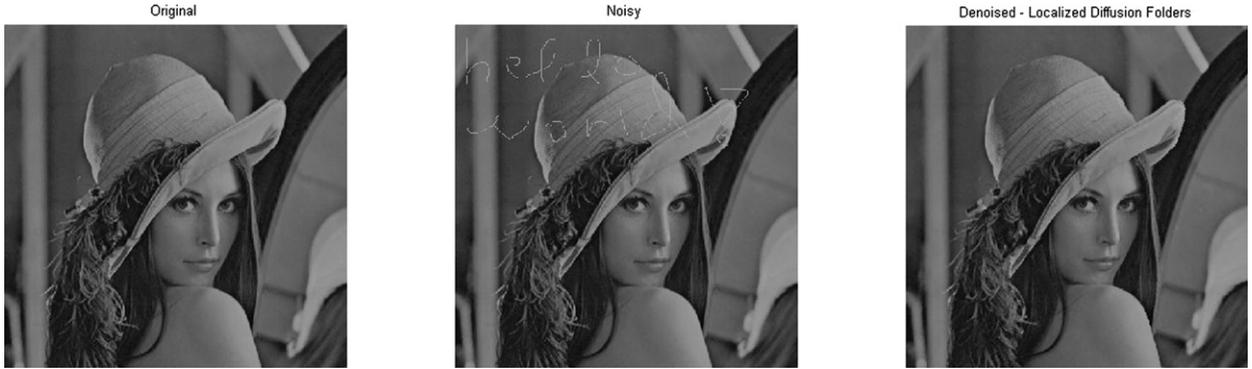


Fig. 4.7. Image restoration results. The left image is the original image. The middle image, which is the damaged one, was obtained by using the pencil tool of an image editor to write some words over the original image. The right image is the output from the application of the LDF denoising algorithm as was described above

A method for denoising using diffusion processes on graphs was described in [33]. Since our LDF based method is most related to this diffusion regularization method, we compare between the performance of both methods. Fig. 4.6 shows the results.

As we can see, the LDF algorithm achieves the best denoising results.

Fig. 4.7 shows another image processing result for restoration (impainting). As we can see, the LDF algorithm restored the damaged image successfully.

5. Conclusions

In this paper, we presented a method for hierarchical clustering via the LDF methodology. The result of this clustering method is a bottom-up hierarchical data clustering while each level in the hierarchy contains LDF or DF from the lower levels. This methodology preserves the local neighborhood of each data point while eliminating noisy and spurious connections between distinct data points and areas in the graph. The performance of the algorithms was demonstrated on real data and was compared with state-of-the-art methods. In most cases, our proposed method outperformed the compared methods. In some cases, when we added noise to the dataset, all the compared methods failed to cluster the data while the LDF succeeded.

Acknowledgments

We would like to thank Ronald R. Coifman for many fruitful discussions and inspirational ideas.

Appendix A. Proof that (A, d) is a metric space

In order to prove that (A, d) is a metric space, we have to show that for any $a_i, a_j, a_l \in A$ the following conditions hold:

- (i) Non-negativity: $d(a_i, a_j) \geq 0$;
- (ii) Identity: $d(a_i, a_j) = 0$ iff $a_i = a_j$;
- (iii) Symmetry: $d(a_i, a_j) = d(a_j, a_i)$;
- (iv) Triangle inequality: $d(a_i, a_j) \leq d(a_i, a_l) + d(a_l, a_j)$.

Conditions (i)–(iii) are satisfied by the definition of d . We prove that condition (iv) is satisfied by contradiction. Assume to the contrary that $d(a_i, a_j) > d(a_i, a_l) + d(a_l, a_j)$. Since d is non-negative then there are two cases to consider: (1) $d(a_i, a_j) = \frac{1}{2}$ or (2) $d(a_i, a_j) = 1$.

- (1) If $d(a_i, a_j) = \frac{1}{2}$ then $d(a_i, a_l) = 0$ and $d(a_l, a_j) = 0$. However, if $d(a_i, a_l) = 0$ and $d(a_l, a_j) = 0$ then $a_i = a_l$ and $a_l = a_j$ (identity). Therefore, we conclude that $a_i = a_j = a_l$ and $d(a_i, a_j) = 0$, contradicting our assumption that $d(a_i, a_j) = \frac{1}{2}$;
- (2) If $d(a_i, a_j) = 1$, then there are three cases to consider: (a) $d(a_i, a_l) = 0$ and $d(a_l, a_j) = 0$ or (b) $d(a_i, a_l) = \frac{1}{2}$ and $d(a_l, a_j) = 0$ or (c) $d(a_i, a_l) = 0$ and $d(a_l, a_j) = \frac{1}{2}$.
 - (a) If $d(a_i, a_l) = 0$ and $d(a_l, a_j) = 0$ then we conclude that $d(a_i, a_j) = 0$ (identity), contradicting our assumption that $d(a_i, a_j) = 1$;
 - (b) If $d(a_i, a_l) = \frac{1}{2}$ and $d(a_l, a_j) = 0$ then since $a_l = a_j$ (identity) we conclude that $d(a_i, a_j) = \frac{1}{2}$, contradicting our assumption that $d(a_i, a_j) = 1$;

(c) Similarly, if $d(a_i, a_l) = 0$ and $d(a_l, a_j) = \frac{1}{2}$ then we conclude that $d(a_i, a_j) = \frac{1}{2}$, contradicting our assumption that $d(a_i, a_j) = 1$.

Therefore, the triangle inequality (condition (iv)) holds and (A, d) is a metric space. Moreover, as we can see in the above proof, the following condition holds as well:

(v) Strong inequality: $d(a_i, a_j) \leq \max\{d(a_i, a_l), d(a_l, a_j)\}$.

Hence, (A, d) is an ultrametric space [34].

Appendix B. Proof that (A, d_μ) is a metric space

As for (A, d) , in order to prove that (A, d_μ) is a metric space, we have to show that for any $a_i, a_j, a_l \in A$ the following conditions hold:

- (i) Non-negativity: $d_\mu(a_i, a_j) \geq 0$;
- (ii) Identity: $d_\mu(a_i, a_j) = 0$ iff $a_i = a_j$;
- (iii) Symmetry: $d_\mu(a_i, a_j) = d_\mu(a_j, a_i)$;
- (iv) Triangle inequality: $d_\mu(a_i, a_j) \leq d_\mu(a_i, a_l) + d_\mu(a_l, a_j)$.

Satisfying condition (i): due to the non-negativity property of d , $d_k(a_i, a_j) \geq 0$ for any k . Therefore, $d_\mu(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) \geq 0$. Hence, (A, d_μ) satisfies the non-negativity condition.

Satisfying condition (ii): first we prove the left side of the identity. Due to the non-negativity property of d , $d_k(a_i, a_j) \geq 0$ for any k . Therefore, if $\frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) = 0$ then $d_k(a_i, a_j) = 0$ for any k . Due to the identity property of d , $a_i = a_j$ for any k . Hence, if $d_\mu(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) = 0$ then $a_i = a_j$.

Now we prove the right side of the identity. Since $a_i = a_j$ then for any k , $d_k(a_i, a_j) = 0$. Therefore, $d_\mu(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) = 0$. Hence, (A, d_μ) satisfies the identity condition.

Satisfying condition (iii): due to the symmetry property of d , $d_k(a_i, a_j) = d_k(a_j, a_i)$ for any k . Therefore, $d_\mu(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) = \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_j, a_i) = d_\mu(a_j, a_i)$. Hence, (A, d_μ) satisfies the symmetry condition.

Satisfying condition (iv): due to the triangle inequality property of d , $d_k(a_i, a_j) \leq d_k(a_i, a_l) + d_k(a_l, a_j)$ for any k . Therefore,

$$\begin{aligned} d_\mu(a_i, a_j) &= \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_j) \\ &\leq \frac{(d_1(a_i, a_l) + d_1(a_l, a_j)) + \dots + (d_{|\tilde{D}^{(t)}|}(a_i, a_l) + d_{|\tilde{D}^{(t)}|}(a_l, a_j))}{|\tilde{D}^{(t)}|} \\ &= \frac{(d_1(a_i, a_l) + \dots + d_{|\tilde{D}^{(t)}|}(a_i, a_l)) + (d_1(a_l, a_j) + \dots + d_{|\tilde{D}^{(t)}|}(a_l, a_j))}{|\tilde{D}^{(t)}|} \\ &= \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_i, a_l) + \frac{1}{|\tilde{D}^{(t)}|} \sum_{k=1}^{|\tilde{D}^{(t)}|} d_k(a_l, a_j) \\ &= d_\mu(a_i, a_l) + d_\mu(a_l, a_j). \end{aligned}$$

Hence, (A, d_μ) satisfies the triangle inequality condition and (A, d_μ) is a metric space.

References

- [1] A. Singer, H. Wu, Orientability and diffusion maps, Appl. Comput. Harmon. Anal. 31 (1) (2011) 44–58.
- [2] R.R. Coifman, S. Lafon, Diffusion maps, Appl. Comput. Harmon. Anal. 21 (1) (2006) 5–30.
- [3] F.R.K. Chung, Spectral Graph Theory, CBMS Reg. Conf. Ser. Math., vol. 92, 1997.
- [4] G. David, Anomaly detection and classification via diffusion processes in hyper-networks, Ph.D. thesis, School of Computer Science, Tel-Aviv University, 2009.
- [5] R.R. Coifman, M. Maggioni, S.W. Zucker, I.G. Kevrekidis, Geometric diffusions for the analysis of data from sensor networks, Curr. Opin. Neurobiol. 15 (2005).
- [6] G. Gan, J.W.C. Ma, Data Clustering: Theory, Algorithms, and Applications, SIAM, 2007.
- [7] J. Macqueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, Berkeley, CA, 1967, pp. 281–297.
- [8] T. Zhang, R. Ramakrishnan, M. Livny, Birch: An efficient data clustering method for very large databases, SIGMOD Rec. 25 (1996) 103–114.

- [9] S. Guha, R. Rastogi, K. Shim, Cure: An efficient clustering algorithms for large databases, in: *ACM SIGMOD International Conference on Management of Data*, Seattle, WA, 1998, pp. 73–84.
- [10] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial database with noise, in: *International Conference on Knowledge Discovery in Databases and Data Mining (KDD-96)*, AAAI Press, Portland, Oregon, 1996, pp. 226–231.
- [11] L. Kaufman, P. Rousseeuw, *Finding Groups in Data – An Introduction to Cluster Analysis*, Wiley, New York, 1990.
- [12] G. David, A. Averbuch, Spectralcat: Categorical spectral clustering of numerical and nominal data, *Pattern Recogn.* 45 (1) (2012) 416–433.
- [13] Z. Huang, Extensions to the k-means algorithm for clustering large data sets with categorical values, *Data Min. Knowl. Discov.* 2 (3) (1998) 283–304.
- [14] V. Ganti, J. Gehrke, R. Ramakrishnan, Cactus: Clustering categorical data using summaries, in: *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, San Diego, CA, USA, 1999, pp. 73–83.
- [15] A. Ben-Hur, D. Horn, H. Siegelmann, V. Vapnik, Support vector clustering, *J. Mach. Learn. Res.* 2 (2001) 125–137.
- [16] M. Girolami, Mercer kernel based clustering in feature space, *IEEE Trans. Neural Networks* 13 (4) (2002) 780–784.
- [17] R. Zhang, A. Rudnicky, A large scale clustering scheme for kernel k-means, in: *Proceedings of the 16th International Conference on Pattern Recognition*, Quebec City, Canada, 2002, pp. 289–292.
- [18] J. Couto, Kernel k-means for categorical data, *Adv. Intell. Data Anal.* VI (3646) (2005) 46–56.
- [19] N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee, M. Verleysen, Width optimization of the Gaussian kernels in radial basis function networks, in: *Proceedings of the European Symposium on Artificial Neural Networks, ESANN*, 2002, pp. 425–432.
- [20] N. Benoudjit, M. Verleysen, On the kernel widths in radial-basis function networks, *Neural Process. Lett.* 18 (2003) 139–154.
- [21] V.D. Sanchez, On the number and the distribution of rbf centers, *Neurocomputing* 7 (1995) 197–202.
- [22] S. Chen, S. Billings, Neural networks for nonlinear dynamic system modelling and identification, *Internat. J. Control* 56 (1992) 319–346.
- [23] M. Orr, Introduction to radial basis function networks, 1996, <http://www.anc.ed.ac.uk/rbf/papers/intro.ps>.
- [24] J. Park, I. Sandberg, Universal approximation using radial basis function networks, *Neural Comput.* 3 (1991) 246–257.
- [25] S. Haykin, *Neural Networks: A Comprehensive Foundation*, second ed., Prentice Hall, 1998.
- [26] M. Verleysen, K. Hlavackova, Learning in rbf networks, in: *International Conference on Neural Networks (ICNN)*, 1996, pp. 199–204.
- [27] A. Saha, J. Keeler, Algorithms for better representation and faster learning in radial basis function networks, *Adv. Neural Inf. Process. Syst.*, vol. 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1990.
- [28] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, *Neural Comput.* 1 (1989) 281–294.
- [29] F. Aurenhammer, Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Comput. Surv.* 23 (1991) 345–405.
- [30] M. Frechet, Sur quelques points du calcul fonctionnel, *Rend. Circ. Mat. Palermo* 22 (1906) 1–74.
- [31] C. Blake, C. Merz, *UCI Repository of Machine Learning Databases*, University of California, Dept. Information and Computer Science, Irvine, CA, USA, 1998, <http://www.ics.uci.edu/mllearn/MLRepository.html>.
- [32] R.A. Fisher, The use of multiple measurements in taxonomic problems, *Ann. Eugenics* 7 (1936) 179–188.
- [33] A.D. Szlam, M. Maggioni, R.R. Coifman, Regularization on graphs with function-adapted diffusion processes, *J. Mach. Learn. Res.* 9 (2008) 1711–1739.
- [34] I. Kaplansky, *Set Theory and Metric Spaces*, AMS Chelsea Publishing, 1977.