RC 22488 (W0206-072) June 14, 2002
Computer Science

IBM Research Report

Within-cluster adaptive metric for clustering

Carlotta Domeniconi, D Gunopulos, Sheng Ma
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Within-Cluster Adaptive Metric for Clustering

Carlotta Domeniconi   Dimitrios Gunopulos
Computer Science Department
University of California, Riverside, CA 92521
{carlotta,dg}@cs.ucr.edu

Sheng Ma
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
shengma@us.ibm.com

Abstract

The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. Clustering suffers from the curse of dimensionality. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may be ineffective. We introduce an algorithm that discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques. Our method associates to each cluster a weight vector, whose values give information of the degree of relevance of features for each set in the partition. We formally prove that our algorithm converges, and experimentally demonstrate the gain in perfomance we achieve with our method.

1 Introduction

The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It has been studied extensively in statistics [3], machine learning [6, 11], and database communities [12, 8, 14].

Given a set of multi-dimensional data, (partitional) clustering finds a partition of the points into clusters such that the points within a cluster are more similar to each other than to points in different clusters. The popular K-means or K-medoids methods compute one representative point per cluster, and assign each object to the cluster with the closest representative, so that the sum of the squared differences between the objects and their representatives is minimized. Finding a set of representative vectors for clouds of multi-dimensional data is an important issue in data compression, signal coding, pattern classification, and function approximation tasks.

Clustering suffers from the curse of dimensionality problem in high dimensional spaces. In high dimensional spaces, it is highly likely that, for any given pair of points within the same cluster, there exist at least a few dimensions on which the points are far apart from each other. It is not meaningful to look for clusters in such a high dimensional space as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may be ineffective.

Furthermore, several clusters may exist in different subspaces, comprised of different combinations of features. In many real world problems, in fact, some points are correlated with respect to a given set of dimensions, and others are correlated with respect to different dimensions. Each dimension could be relevant to at least one of the clusters.

The problem of high dimensionality could be addressed by requiring the user to specify a subspace (i.e., subset of dimensions) for cluster analysis. However, the identification of subspaces by the user is an error-prone process. More importantly, correlations that identify clusters in the data are likely not to be known by the user. Indeed, we desire such correlations, and induced subspaces, to be part of the findings of the clustering process itself.

An alternative solution to high dimensional settings consists in reducing the dimensionality of the input space. Traditional feature selection algorithms select certain dimensions in advance. Methods such as Principal Component Analysis (PCA) (or Karhunen-Loeve transformation) [7, 9] transform the original input space into a lower dimensional space by constructing dimensions that are linear combinations of the given features, and are ordered by nonincreasing variance. While PCA may succeed in reducing the dimensionality, it has major drawbacks. The new dimensions can be difficult to interpret, making it hard to understand clusters in relation to the original space. Furthermore, all global dimensionality reduction techniques

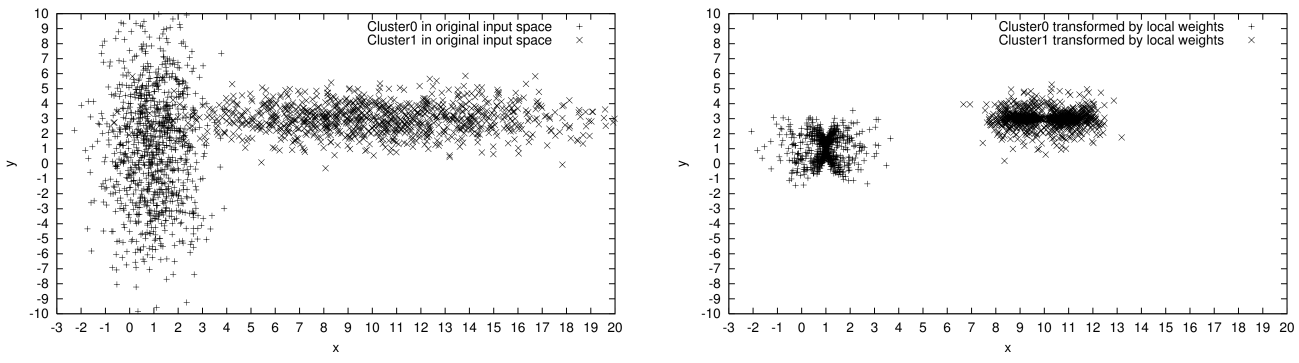

Figure 1: (Left) Clusters in original input space. (Right) Clusters transformed by local weights.

(like PCA) are not effective in identifying clusters that may exist in different subspaces. In this situation, in fact, since data across clusters manifest different correlations with features, it may not always be feasible to prune off too many dimensions without incurring a loss of crucial information. This is because each dimension could be relevant to at least one of the clusters.

These limitations of global dimensionality reduction techniques suggest that, to capture the local correlations of data, a proper feature selection procedure should operate locally in input space. Local feature selection allows to embed different distance measures in different regions of the input space; such distance metrics reflect local correlations of data. In this paper we propose a *soft* feature selection procedure that assigns (local) weights to features according to the local correlations of data along each dimension. Dimensions along which data are loosely correlated receive a small weight, that has the effect of elongating distances along that dimension. Features along which data are strongly correlated receive a large weight, that has the effect of constricting distances along that dimension. Figure 1 gives a simple example. The left plot depicts two clusters of data elongated along the x and y dimensions. The right plot shows the same clusters, where within-cluster distances between points are computed using the respective local weights generated by our algorithm (GenProClus). The weight values reflect local correlations of data, and reshape each cluster as a *dense spherical cloud*. This directional local reshaping of distances better separates clusters, and allows for the discover of different patterns in different subspaces of the original input space.

1.1 Our Contribution

The contributions of this paper are as follows:

1. We formalize the problem of finding different clusters in different subspaces. Our algorithm discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques.

2. The output of our algorithm is twofold. It provides a partition of the data, so that the points in each set of the partition constitute a cluster. In addition, each set is associated with a weight vector, whose values give information of the degree of relevance of features for each partition.

3. We formally prove that our algorithm converges to a local minimum of the associated error function, and experimentally demonstrate the gain in perfomance we achieve with our method.

2 Related Work

Local dimensionality reduction approaches for the purpose of efficiently indexing high dimensional spaces have been recently discussed in the database literature [10, 5, 13]. Applying global dimensionality reduction techniques when data are not globally correlated can cause significant loss of distance information, resulting in a large number of false positives and hence a high query cost. The general approach adopted by the authors is to find local correlations in the data, and perform dimensionality reduction on the locally correlated clusters individually. For example, in [5], the authors first construct spacial clusters in the original input space using a simple tecnique that resembles K-means. Principal component analysis is then performed on each spatial cluster individually to obtain the principal components.

In general, the efficacy of these methods depends on how the clustering problem is addressed in the first place in the original feature space. A potential serious problem with such techniques is the lack of data to locally perform PCA on each cluster to derive the principal components. Moreover, for clustering purposes, the new dimensions may be difficult to interpret, making it hard to understand clusters in relation to the original space.

The problem of finding different clusters in different subspaces of the original input space has been addressed in [2]. The authors use a density based approach to identify clusters. The algorithm (CLIQUE) proceeds from lower to higher dimensionality subspaces and discovers dense regions in each subspace. To approximate the density of the points, the input space is partitioned into cells by dividing each dimension into the same number ξ of equal length intervals. For a given set of dimensions, the cross product of the corresponding intervals (one for each dimension in the set) is called a *unit* in the respective subspace. A unit is dense if the number of points it contains is above a given threshold τ. Both ξ and τ are parameters defined by the user. The algorithm finds all dense units in each k-dimensional subspace by building from the dense units of $(k-1)$-dimensional subspaces, and then connects them to describe the clusters as union of maximal rectangles.

While the work in [2] successfully introduces a methodology for looking at different subspaces for different clusters, it does not compute a partitioning of the data into disjoint groups. The reported dense regions largely overlap, since for a given dense region all its projections on lower dimensionality subspaces are also dense, and they all get reported. On the other hand, for many applications such as customer segmentation and trend analysis, a partition of the data is desirable since it provides a clear interpretability of the results.

The problem of finding different clusters in different subspaces is also addressed in [1]. The proposed algorithm (PROjected CLUStering) seeks subsets of dimensions such that the points are closely clustered in the corresponding spanned subspaces. Both the number of clusters and the average number of dimensions per cluster are user-defined parameters. PROCLUS starts with choosing a random set of medoids, and then progressively improves the quality of medoids by performing an iterative hill climbing procedure that discards the 'bad' medoids from the current set. In order to find the set of dimensions that matter the most for each cluster, the algorithm selects the dimensions along which the points have the smallest average distance from the current medoid. The authors do not prove that the algorithm converges to the optimality criterion they choose.

Our method (that we call GENeralized PROjected CLUStering) can be seen as a generalization of PROCLUS. Our method does not require to specify the average number of dimensions to be kept per cluster. For each cluster, in fact, *all* features are taken into consideration, but properly weighted. The PROCLUS algorithm is more prone to loss of information if the number of dimensions is not properly chosen. For example, if data of two clusters in two dimensions are distributed as in Figure 2, PROCLUS may find that feature x is the most important for cluster 0, and feature y is the most important for cluster 1. But projecting cluster 1 along the y dimension doesn't allow to properly separate points of the two clusters. We avoid this problem by keeping both dimensions for both clusters, and properly weighting distances along each feature within each cluster.

3 Problem Statement

We define what we call *weighted cluster*. Consider a set of points in some space of dimensionality N. A *weighted cluster* C is a subset of data points, together with a vector of weights $\mathbf{w} = (w_1, \ldots, w_N)$, such that the points in C are closely clustered according to the L_2 norm distance weighted using $\mathbf{w}$. The component w_j measures the degree of correlation of points in C along feature j. The problem becomes now how to estimate the weight vector $\mathbf{w}$ for each cluster in the data set.

In this setting, the concept of *cluster* is not based only on points, but also involves a weighted distance metric, i.e., clusters are been discovered in spaces transformed by $\mathbf{w}$. Each cluster is associated with its own $\mathbf{w}$, that reflects the correlation of points in the cluster itself. The effect of $\mathbf{w}$ is to transform distances so that the associated cluster is reshaped into a dense hypersphere of points separated from other data.

In traditional clustering, the partition of a set of points is induced by a set of *representative* vectors, also called *centroids* or *centers*. The partition induced by discovering weighted clusters is formally defined as follows.

Definition: Given a set S of D points $\mathbf{x}$ in N-dimensional Euclidean space, a set of k centers $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$, $\mathbf{c}_j \in \Re^N$, $j = 1, \ldots, k$, coupled with a set of corresponding weight vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$, $\mathbf{w}_j \in \Re^N$, $j = 1, \ldots, k$, partition S into k sets $\{S_1, \ldots, S_k\}$:

$$S_j = \{\mathbf{x} | (\sum_{i=1}^{N} w_{ji}(x_i - c_{ji})^2)^{1/2} < (\sum_{i=1}^{N} w_{li}(x_i - c_{li})^2)^{1/2}, l \neq j\} \tag{1}$$

The set of centers and weights is *optimal* with respect to the Euclidean norm, if they minimize the error measure:

$$E_1(C, W) = \sum_{j=1}^{k} \sum_{i=1}^{N} w_{ji} e^{(X_j - X_{ji})} \tag{2}$$

subject to the constraints $\sum_{i=1}^{N} w_{ji}^2 = 1 \ \forall j$. C and W are $(N \times k)$ matrices whose column vectors are $\mathbf{c}_j$ and $\mathbf{w}_j$ respectively, i.e. $C = [\mathbf{c}_1 \ldots \mathbf{c}_k]$ and $W = [\mathbf{w}_1 \ldots \mathbf{w}_k]$. X_j and X_{ji} are defined as follows: $X_j = \frac{1}{|S_j|}(\max_l(\sum_{\mathbf{x} \in S_j}(c_{jl} - x_l)^2))$ and $X_{ji} = \frac{1}{|S_j|}(\sum_{\mathbf{x} \in S_j}(c_{ji} - x_i)^2)$. X_{ji} represents the average distance from the centroid $\mathbf{c}_j$ of points in cluster j along dimension i, and X_j is the largest of such average distances $\forall i$.

The minimization of the error function E_1 as defined in (2) has a specific geometric interpretation. The optimal solution aims to minimize, for each cluster, the (exponential of the) discrepancy between the largest spread (X_j) of data along each dimension, and each of such spreads (X_{ji}). As a result, in the space transformed by optimal weights, the corresponding cluster has the shape of a hypersphere well separated from other data. The exponential function in (2) has the effect of making the weights w_{ji} more sensitive to the discrepancy $(X_j - X_{ji})$, and therefore to changes in local feature relevance. As a consequence, clusters are better separated in the transformed spaces, and large performance improvements can be achieved as also demonstrated with our experimental results.

In the following we present an algorithm that finds a solution (set of centers and weights) that is a local minimum of the error function (2).

4 Generalized Projected Clustering Algorithm

We start with *well-scattered* points in D as the k centroids [1], and initially set the weights' values to 1. We progressively improve the quality of the centroids and of the weights by investigating the space near the centers, in order to estimate the dimensions that matter the most, i.e. the dimensions along which local data are mostly correlated. Specifically, we proceed as follows.

Given the initial centroids $\mathbf{c}_j$, for $j = 1, \ldots, k$, we compute the corresponding sets S_j as defined in (1), where $w_{ji} = 1 \ \forall j$ and $\forall i$. We then compute the average distance along each dimension from the points in S_j to $\mathbf{c}_j$. Let X_{ji} denote this average distance along dimension

i, and let X_j the largest average distance among the N dimensions for cluster j. The smaller X_{ji} is, the larger is the correlation of points along dimension i. Thus, the difference $X_j - X_{ji}$ gives us a value that is proportional to the amount of correlation of points along feature i. Let $X'_{ji} = X_j - X_{ji}$. We use the value X'_{ji} in an exponential weighting scheme to credit weights to features (and to clusters):

$$w_{ji} = exp(h \times X'_{ji}) / (\sum_{l=1}^{N} (exp(h \times 2 \times X'_{jl})))^{1/2} \tag{3}$$

where h is a parameter that can be chosen to maximize (minimize) the influence of X'_{ji} on w_{ji}. When $h = 0$ we have $w_{ji} = 1/N$, thereby ignoring any difference between the X'_{ji}. On the other hand, when h is large a change in X'_{ji} will be exponentially reflected in w_{ji}. We empirically determine the value of h through cross-validation in our experiments. The exponential weighting is more sensitive to changes in local feature relevance [4] and gives rise to better performance improvement. In fact, it is more stable because it prevents distances from extending infinetely in any direction, i.e., zero weight. This, however, can occur when either linear or quadratic weighting is used.

These weights w_{ji} enable to elongate distances along less important dimensions, i.e. dimensions along which points are loosely correlated, and, at the same time, to constrict distances along the most influential ones, i.e. features along which points are strongly correlated. Note that the technique is centroid-based because weightings depend on the centroid.

The computed weights are used to update the sets S_j, and therefore the centroids' coordinates. The procedure is iterated until convergence is reached, i.e. no change in centers' coordinates is observed.

The resulting algorithm, that we call GenProClus, is summarized in the following.

Input: Set D of points $\mathbf{x} \in R^N$, and the number of clusters k.

1. Start with k initial centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$;

2. Set $w_{ji} = 1$, for each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$ and each feature $i = 1, \ldots, N$;

3. For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each data point $\mathbf{x}$:

    - Set $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$,
      where $WDist(\mathbf{c}_l, \mathbf{x}) = (\sum_{i=1}^{N} w_{li}(c_{li} - x_i)^2)^{1/2}$;

4. **Compute new weights**. For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each feature i:

    - Set $X_{ji} = \sum_{\mathbf{x} \in S_j} (c_{ji} - x_i)^2 / |S_j|$, where $|S_j|$ is the cardinality of set S_j (X_{ji} represents the average distance of points in S_j from $\mathbf{c}_j$ along feature i);
    - Set $X_j = \max_i X_{ji}$;
    - Set $X'_{ji} = X_j - X_{ji}$;
    - Set $w_{ji} = exp(h \times X'_{ji}) / \sum_{l=1}^{N} (exp(h \times 2 \times X'_{jl}))^{1/2}$;

5. For each centroid $\mathbf{c}_j$, $j = 1, \ldots, k$, and for each data point $\mathbf{x}$:

    - Recompute $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$;

6. **Compute new centroids**. Set $\mathbf{c}_j = \frac{\sum_{\mathbf{x}} \mathbf{x} 1_{S_j}(\mathbf{x})}{\sum_{\mathbf{x}} 1_{S_j}(\mathbf{x})}$, for each $j = 1, \ldots, k$, where $1_S(.)$ is the indicator function of set S;

7. Iterate 3,4,5 until convergence (i.e., no change in centroids' coordinates)

5 Convergence of the GenProClus Algorithm

To formally prove convergence of the GenProClus algorithm we need an error function that is differentiable with respect to both w_{ji} and c_{ji}. We observe that the error measure in (2), while sound in theory, is not differentiable due to the definition of X_j in terms of a max function. We solve this problem by substituting X_j with a value X that measures the largest spread of the projections of data in any dimensions. We observe that X is constant given the data set, and therefore does not depend on w_{ji} or c_{ji}. X provides an upper bound for all X_j, $j = 1, \ldots, k$. Thus, the resulting error function obeys the same principle that motivates the original error measure (2).

The resulting error function to be considered is:

$$E_2(C, W) = \sum_{j=1}^{k} \sum_{i=1}^{N} w_{ji} e^{(X - X_{ji})} \tag{4}$$

where $X = \frac{1}{D}(\max_l(\sum_{\mathbf{x} \in S}(m_l - x_l)^2))$, $\mathbf{m} = \frac{1}{D}\sum_{\mathbf{x} \in S} \mathbf{x}$, and X_{ji} is defined as before. Our objective becomes the minimization of (4) subject to the constraints $\sum_i {w_{ji}}^2 = 1 \; \forall j$. We can solve this constrained optimization problem by introducing the Lagrange multipliers λ_j (one for each constraint), and minimizing the resulting (unconstrained now) error function

$$E(C,W) = \sum_{j=1}^{k}\sum_{i=1}^{N} w_{ji} e^{(X-X_{ji})} + \sum_{j=1}^{k} \lambda_j (1 - \sum_{i=1}^{N} w_{ji}^2) \tag{5}$$

We prove the following theorem.

Theorem. The GenProClus algorithm converges to a local minimum of the error function (5).

Proof. For fixed c_{ji} and x_i, we compute the optimal w_{ji} by setting $\frac{\partial E}{\partial w_{ji}} = 0$ and $\frac{\partial E}{\partial \lambda_j} = 0$. We obtain:

$$\frac{\partial E}{\partial w_{ji}} = e^{X-X_{ji}} - 2\lambda_j w_{ji} = 0 \tag{6}$$

$$\frac{\partial E}{\partial \lambda_j} = 1 - \sum_{i=1}^{N} w_{ji}^2 = 0 \tag{7}$$

Solving equation (6) with respect to w_{ji} we obtain $w_{ji} = \frac{e^{X-X_{ji}}}{2\lambda_j}$. Substituting this expression in equation (7), and solving with respect to λ_j we obtain $\lambda_j = \frac{1}{2}(\sum_{i=1}^{N} e^{2(X-X_{ji})})^{1/2}$. Thus, the optimal w_{ji} is

$$w_{ji} = \frac{e^{X-X_{ji}}}{(\sum_{l=1}^{N} e^{2(X-X_{jl})})^{1/2}}$$

as in step 4 of the GenProClus algorithm.

For fixed j and w_{ji}, we compute the optimal c_{ji} by setting $\frac{\partial E}{\partial c_{ji}} = 0$. We obtain:

$$\frac{\partial E}{\partial c_{ji}} = w_{ji} e^{(X-X_{ji})} (-\frac{2}{|S_j|} \sum_{\mathbf{x} \in S_j} (c_{ji} - x_i)) = 0 \tag{8}$$

Solving equation (8) with respect to c_{ji} we obtain:

$$c_{ji} = \frac{1}{|S_j|} \sum_{\mathbf{x} \in S_j} x_i$$

as in step 6 of the GenProClus algorithm.

This shows that at each iteration the GenProClus algorithm computes the optimal w_{ji}s and c_{ji}s for the given partition of data. Thus at each iteration points are re-distributed among clusters according to the optimal w_{ji}s and c_{ji}s. Therefore the value of E decreases after each iteration, and the algorithm converges in a finite number of steps to a local minimum of E.

6 Experimental Evaluation

In our experiments we have designed six different simulated data sets. Clusters are distributed according to multivariate gaussians with different mean and standard deviation vectors. We have tested problems with two and three clusters up to ten dimensions. For each problem, we have generated 10 training data sets, and for each of them an independent test set. In the following we report performance results obtained via 10-fold cross-validation comparing GenProClus and K-means algorithms. The k centroids for both algorithms are initialized by choosing well-scattered points among the given data. To facilitate the interpretation of weight values, we require that $\sum_i w_{ji} = 1 \ \forall j$ in our experiments, by properly adjusting the normalization factor of the weighting scheme (3).

6.1 The Problems

1. **Example1**. This data set consists of $n = 2$ attributes and $J = 2$ clusters. Data for one cluster are generated from a multivariate normal distribution with mean vector $(1, 1)$ and standard deviations $(1, 4)$. Data for the other cluster are generated from a normal distribution with mean vector $(10, 3)$ and standard deviations $(4, 1)$. Figure 2 shows the distributions of data for the two clusters. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 1.

2. **Example2**. This data set consists of $n = 3$ attributes and $J = 2$ clusters. One cluster is drawn from a multivariate normal distribution with mean vector $(1, 1, 1)$, and standard deviations $(1, 4, 1)$. The second cluster is drawn again form a multivariate normal distribution with mean vector $(5, 5, 1)$, and standard deviations $(1, 1, 4)$. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 1.

3. **Example3**. The data set consists of $n = 3$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, 1, 1)$ are $(1, 4, 1)$ respectively. For the other cluster the vectors are $(3, 1, 1)$

and $(1,1,4)$. Figures 3-4 shows the two clusters projected in $x-y$, $x-z$, and $y-z$ spaces respectively. Table 1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

4. **Example4**. The data set consists of $n=5$ input features and $J=2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1,1,1,1,1)$ and $(1,4,1,4,1)$ respectively. For the other cluster the vectors are $(5,1,1,1,1)$ and $(4,1,1,1,4)$. Table 1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

5. **Example5**. The data set consists of $n=2$ input features and $J=3$ clusters. All three clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(2,0)$ are $(4,1)$ respectively. For the second cluster the vectors are $(10,0)$ and $(1,4)$, and for the third are $(18,0)$ and $(4,1)$. Table 1 shows the results for this problem. We generated 60000 data points, and performed 10-fold cross-validation with 30000 training data and 30000 testing data.

6. **Example6**. The data set consists of $n=10$ input features and $J=2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1,\dots,1)$ and $(1,5,1,5,1,5,1,5,1,5)$ respectively. For the other cluster the vectors are $(5,1,\dots,1)$ and $(5,1,5,1,5,1,5,1,5,1)$. Table 1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

Table 1: Average error rates.

	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6
GenProClus	2.7	0.9	7.0	4.8	11.4	0.1
K-Means	11.9	7.2	19.2	35.1	24.2	42.2

Table 2: Average number of iterations.

	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6
GenProClus	5.3	3.9	6.1	5.4	7.2	3.1
K-Means	16.1	10.7	28.5	33.8	16.8	32.7

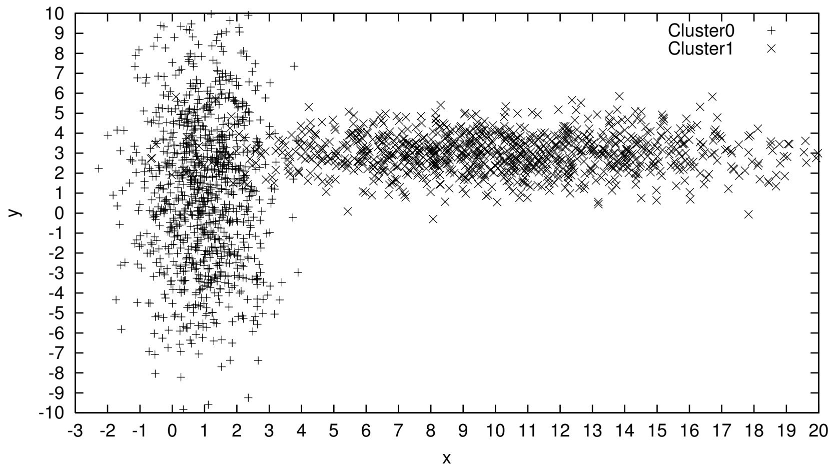

Figure 2: Example1: Distributions of clusters.

Table 3: GenProClus: Confusion matrix for Example1.

	C0 (input)	C1 (input)
C0 (output)	9917	464
C1 (output)	83	9536

Table 4: Kmeans: Confusion matrix for Example1.

	C0 (input)	C1 (input)
C0 (output)	8364	737
C1 (output)	1636	9263

Table 5: GenProClus: Weight values for Example1.

Cluster	Std1	Std2	w_1	w_2
C0	1	4	0.999	0.001
C1	4	1	0.045	0.955

Table 6: GenProClus: Confusion matrix for Example2.

	C0 (input)	C1 (input)
C0 (output)	9895	72
C1 (output)	105	9928

6.1.1 Results

The performance results reported in Table 1 clearly demonstrate the large gain in performance obtained by the GenProClus algorithm against K-means. In particular, the large error rate of K-means (42.2) for the 10 dimensional data set (Example 6) shows how ineffective a distance function that equally use all input features can be in moderately high dimensional spaces. The gain in performance achieved by locally weighting features is huge in this case.

Table 2 shows the average number of iterations performed by each algorithm to achieve convergence. For each problem, the rate of convergence of GenProClus is at least three times faster (except for problem 5 where is 2.3 times faster); for problem 6 is 10 times faster.

To further test the accuracy of the algorithms, for each problem we have computed the

Table 7: Kmeans: Confusion matrix for Example2.

	C0 (input)	C1 (input)
C0 (output)	8578	25
C1 (output)	1422	9975

Table 8: GenProClus: Weight values for Example2.

Cluster	Std1	Std2	Std3	w_1	w_2	w_3
C0	1	4	1	0.40	0.02	0.58
C1	1	1	4	0.33	0.66	0.01

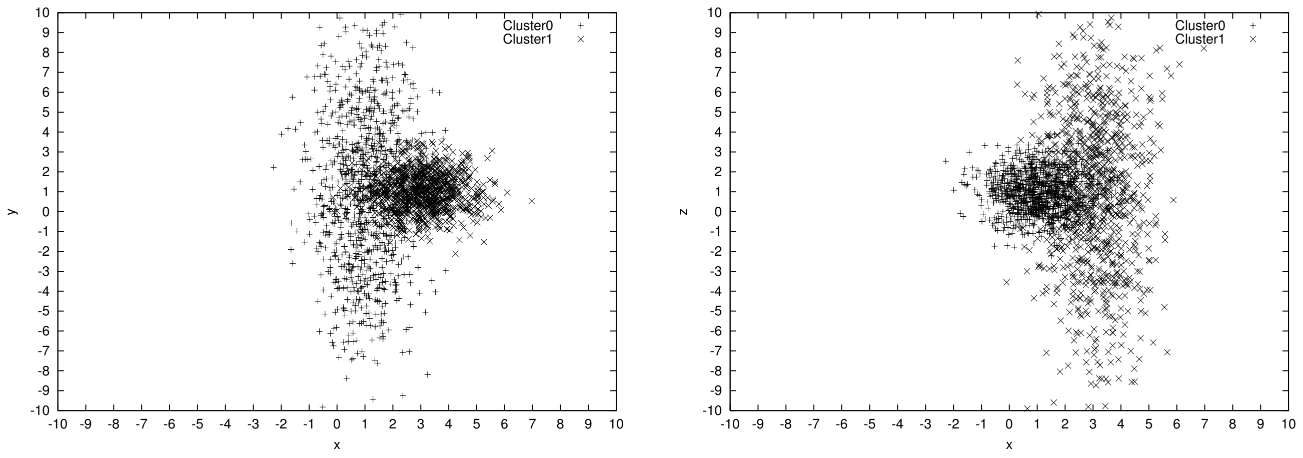

Figure 3: Example3: (Left) Distributions of clusters in x-y space. (Right) Distributions of clusters in x-z space.

confusion matrices. The entry (i, j) in each confusion matrix is equal to the number of points assigned to output cluster i, that were generated as part of input cluster j. We also report the average weight values per cluster obtained over the 10 run conducted in our experiements. Results are reported in Tables 3-20.

Tables 5,8,11,14,17 and 20 show that there is a perfect correspondence between the weight values of each cluster and the correlation patterns of data within the same cluster. This is of great importance for applications that require not only a good partitioning of data, but also information to what features are relevant for each partition. Figures 3 and 4 show the data distributions of the two clusters of Example 3 projected in the $x - y$, $x - z$, and $y - z$ planes, respectively. We observe that data of cluster 0 are closely correlated in the subspace $x - z$, whereas data of cluster 1 are closely correlated in the subspace $x - y$. Table 11 shows that the resulting weight values reflect such local correlations, i.e., larger weights w_1 and w_3 are credited to cluster 0, and larger weights w_1 and w_2 are credited to cluster 1.

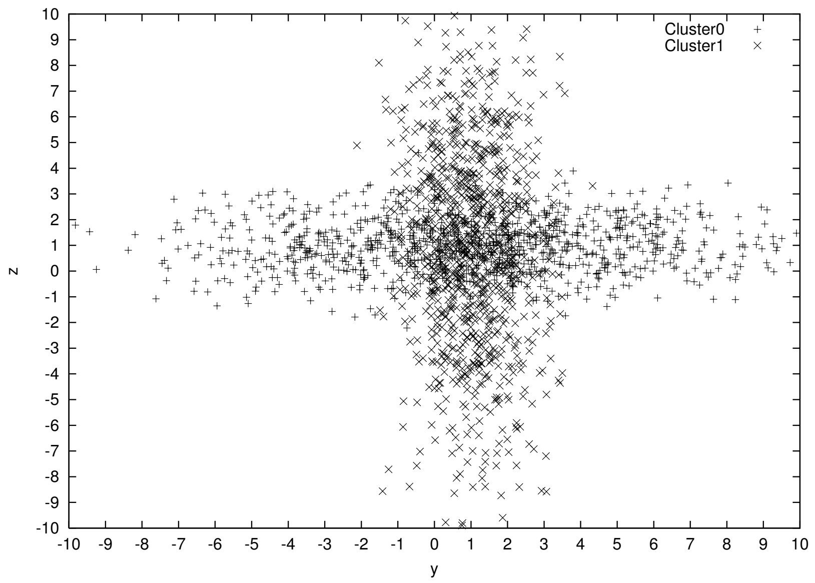

Figure 4: Example3: Distributions of clusters in y-z space.

Table 9: GenProClus: Confusion matrix for Example3.

	C0 (input)	C1 (input)
C0 (output)	9313	705
C1 (output)	687	9295

As expected, the resulting weight values for one cluster depends on the configurations of other clusters as well. If clusters have the same standard deviation along one dimension i, they receive almost identical weights for measuring distances along that feature. This is informative of the fact that feature i is equally relevant for both partitions. On the other hand, weight values are largely differentiated when two clusters have different standard deviation values along the same dimension i, implying different degree of relevance of feature i for the two partitions (see for example Tables 11, 14, and 17).

Table 10: Kmeans: Confusion matrix for Example3.

	C0 (input)	C1 (input)
C0 (output)	7786	1629
C1 (output)	2214	8371

Table 11: GenProClus: Weight values for Example3.

Cluster	Std1	Std2	Std3	w_1	w_2	w_3
C0	1	4	1	0.22	0.01	0.77
C1	1	1	4	0.21	0.78	0.01

7 Conclusions

We have introduced an algorithm to discover clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques. Our experiments show that there is a perfect correspondence between the weight values of each cluster and local correlations of data. We formally prove that our algorithm converges, and experimentally demonstrate the gain in perfomance we achieve with our method. We plan to conduct more extensive experiments using real data and comparisons with other algorithms in our future work.

Table 12: GenProClus: Confusion matrix for Example4.

	C0 (input)	C1 (input)
C0 (output)	9688	654
C1 (output)	312	9346

Table 13: Kmeans: Confusion matrix for Example4.

	C0 (input)	C1 (input)
C0 (output)	7249	4265
C1 (output)	2751	5735

Table 14: GenProClus: Weight values for Example4.

Cluster	Std1	Std2	Std3	Std4	Std5	w_1	w_2	w_3	w_4	w_5
C0	1	4	1	4	1	0.49	0.02	0.05	0.02	0.42
C1	4	1	1	1	4	0.04	0.45	0.05	0.45	0.01

References

[1] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, "Fast Algorithms for Projected Clustering", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998.

[3] P. Arabie and L. J. Hubert, "An overview of combinatorial data analysis". In P. Arabie, L. J. Hubert, and G. D. Soete editors, *Clustering and Classification*, pp. 5-63, World Scientific Pub., 1996.

Table 15: GenProClus: Confusion matrix for Example5.

	C0 (input)	C1 (input)	C2 (input)
C0 (output)	8315	0	15
C1 (output)	1676	10000	1712
C2 (output)	9	0	8273

Table 16: Kmeans: Confusion matrix for Example5.

	C0 (input)	C1 (input)	C2 (input)
C0 (output)	9440	4686	400
C1 (output)	411	3953	266
C2 (output)	149	1361	9334

Table 17: GenProClus: Weight values for Example5.

Cluster	Std1	Std2	w_1	w_2
C0	4	1	0.46	0.54
C1	1	4	0.99	0.01
C2	4	1	0.45	0.55

[4] L. Bottou and V. Vapnik, "Local learning algorithms" *Neural computation*, 4(6):888-900, 1992.

[5] K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: a new approach to indexing high dimensional spaces", *Proceeding of VLDB*, 2000.

[6] P. Cheeseman and J. Stutz, "Bayesian classification (autoclass): theory and results". In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pp. 153-180, AAAI/MIT Press, 1996.

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

[8] M. Ester, H. P. Kriegel, and X. Xu, "A database interface for clustering in large spatial databases", *Proceedings of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining*,

Table 18: GenProClus: Confusion matrix for Example6.

	C0 (input)	C1 (input)
C0 (output)	9992	11
C1 (output)	8	9989

Table 19: Kmeans: Confusion matrix for Example6.

	C0 (input)	C1 (input)
C0 (output)	5933	4372
C1 (output)	4067	5628

Table 20: GenProClus: Weight values for Example6.

Cluster	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
C0	0.21	0.005	0.19	0.005	0.19	0.005	0.19	0.005	0.19	0.01
C1	0.01	0.19	0.005	0.19	0.01	0.20	0.01	0.19	0.005	0.19

1995.

[9] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.

[10] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases", *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2001.

[11] R. S. Michalski and R. E. Stepp, "Learning from observation: Conceptual clustering". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell editors, *Machine Learning: An Artificial Intelligence Approach*, 1:331-363, Morgan Kaufmann, 1983.

[12] R. T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining", *Proceedings of the VLDB conference*, 1994.

[13] A. Thomasian, V. Castello, and C. S. Li, "Clustering and singular value decomposition for approximate indexing in high dimensional spaces", *Proceedings of CIKM*, 1998.

[14] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases", *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1996.