RIC: Parameter-Free Noise-Robust Clustering

10

CHRISTIAN BÖHM
University of Munich
CHRISTOS FALOUTSOS
Carnegie Mellon University
JIA-YU PAN
Google
and
CLAUDIA PLANT
University of Munich

How do we find a *natural* clustering of a real-world point set which contains an unknown number of clusters with different shapes, and which may be contaminated by noise? As most clustering algorithms were designed with certain assumptions (Gaussianity), they often require the user to give input parameters, and are sensitive to noise. In this article, we propose a robust framework for determining a natural clustering of a given dataset, based on the minimum description length (MDL) principle. The proposed framework, *robust information-theoretic clustering (RIC)*, is orthogonal to any known clustering algorithm: Given a preliminary clustering, RIC purifies these clusters from noise, and adjusts the clusterings such that it simultaneously determines the most natural amount and shape (subspace) of the clusters. Our RIC method can be combined with any clustering technique ranging from K-means and K-medoids to advanced methods such as spectral clustering.

This material is based upon work supported by the National Science Foundation under Grants IIS-0209107, SENSOR-0329549, EF-0331657, IIS-0326322, and IIS-0534205. This work is also supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), a partnership of Carnegie Mellon, Lehigh University and the Commonwealth of Pennsylvania's Department of Community and Economic Development (DCED). Additional funding was also provided by donations from Intel, NTT, and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

Authors' addresses: C. Böhm, Institute for Computer Science, University of Munich, Oettingenstrasse 67, D-80538 Munich, Germany; C. Faloutsos, Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213; J.-Y. Pan, Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043; C. Plant (contact author), Institute for Computer Science, University of Munich, Oettingenstrasse 67, D-80538 Munich, Germany; email: plant@dbs.ifi.lmu.de.

In fact, RIC is even able to purify and improve an initial coarse clustering, even if we start with very simple methods. In an extension, we propose a fully automatic stand-alone clustering method and efficiency improvements. RIC scales well with the dataset size. Extensive experiments on synthetic and real-world datasets validate the proposed RIC framework.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; E.4 [**Data**]: Coding and Information Theory—*Data compaction and compression*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Clustering, noise robustness, parameter-free data mining, data summarization

ACM Reference Format:

Böhm, C., Faloutsos, C., Pan, J.-Y., and Plant, C. 2007. RIC: Parameter-free noise-robust clustering. ACM Trans. Knowl. Discov. Data. 1, 3, Article 10 (December 2007), 29 pages. DOI = 10.1145/1297332.1297334 http://doi.acm.org/10.1145/1297332.1297334

1. INTRODUCTION

The problem of clustering has attracted a huge volume of attention for several decades, with multiple books [Hartigan 1975; Van-Rijsbergen 1979], surveys [Murtagh 1983] and papers (X-means [Pelleg and Moore 2000], G-means [Hamerly and Elkan 2003], CLARANS [Ng and Han 1994], CURE [Guha et al. 1998], CLIQUE [Agrawal et al. 1998], BIRCH [Zhang et al. 1996], DBSCAN [Ester et al. 1996], to name a few). Recent interest in clustering has been on finding clusters that have non-Gaussian correlations in subspaces of the attributes, for example, the work of Böhm et al. [2004], Tung et al. [2005], and Aggarwal and Yu [2000]. Finding correlation clusters has diverse applications ranging from spatial databases to bioinformatics. The hard part of clustering is to decide what is a good group of clusters, and which data points to label as outliers and thus ignore for clustering.

For example, in Figure 1, we show a fictitious set of points in 2D. Figure 1(a) shows a grouping of points that most humans would agree is "good:" a Gaussian-like cluster at the left, a line-like cluster at the right, and a few noise points (outliers) scattered throughout. However, typical clustering algorithms like K-means may produce a clustering like the one in Figure 1(b): a bad number of clusters (five in this example) with Gaussian-like shapes, fooled by a few outliers. There are two questions that we try to answer in this work, described as follows:

—*Q1: Goodness.* How can we quantify the goodness of a grouping? We would like a function that will give a good score to the grouping of Figure 1(a) and a bad one to that of Figure 1(b).

—*Q2: Efficiency.* How can we write an algorithm that will produce good groupings efficiently and without getting distracted by outliers?

The contributions of this article directly address the preceding two questions: For the first, we propose to envision the problem of clustering one of compression and we use information-theoretic arguments. The grouping of Figure 1(a) is good because it can succinctly describe the given dataset, with few exceptions:

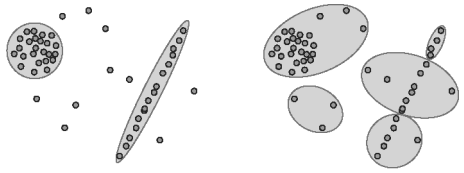


Fig. 1. A fictitious dataset (a) with a good clustering of one Gaussian cluster, one subspace cluster, and noise; and (b) a bad clustering.

The points of the left cluster can be described by their (short) distances from the cluster center; the points on the right line-like cluster can be described by just one coordinate (the location on the line), instead of two; the remaining outliers each need two coordinates, with near-random (and thus uncompressible) values. Our proposal is to measure the goodness of a grouping as the *volume after compression* (VAC): that is, record the bytes to describe the number of clusters k; those to record their type (Gaussian, line-like, or something else from a fixed vocabulary of distributions); those to describe the parameters of each distribution (e.g., mean, variance, covariance, slope, intercept), and then the location of each point, compressed according to the distribution, it belongs to which.

Notice that the VAC criterion does not specify *how* to find a good grouping; it can only say which of two groupings is better. This brings us to the next contribution of this article: We propose to start from a suboptimal grouping (e.g., using K-means, with some arbitrary k). Then, we propose to use two novel algorithms, which we list next.

—*Robust Fitting (RF)*. Instead of the fragile PCA, this algorithm finds low-dimensionality subspace clusters; and

—*Cluster Merging (CM)*. This one can stitch promising clusters together.

We continue fitting and merging, until our VAC criterion reaches a plateau. The aforesaid sketch of our algorithm has a gradient-descent flavor. Notice that we can use *any* and *all* of the known optimization methods, like simulated annealing, genetic algorithms, and everything else that we want: Our goal is to optimize our VAC criterion within the user-acceptable time frame. We propose the gradient-descent version because we believe it strikes a good balance between speed of computation and cluster quality.

1.1 Contributions

The proposed method, RIC, answers both questions that we stated earlier: For cluster quality, it uses the information-theoretic VAC criterion; for searching, it uses the two new algorithms (robust fitting and cluster merging). The resulting method has the following advantages.

—It is fully automatic, that is, no difficult or sensitive parameters must be selected by the user.

—It returns a natural partitioning of the dataset, thanks to the intuitive information-theoretic principle of maximizing the data compression.
—It can detect clusters beyond Gaussians: clusters in full-dimensional data space as well as clusters in axis-parallel subspaces (so-called subspace clusters) and in arbitrarily oriented subspaces (correlation clusters), in addition to combinations and mixtures of clusters of all different types during one single run of the algorithm.
—It can assign model distribution functions such as uniform, Gaussian, Laplacian (etc.) distribution to the different subspace coordinates and thus gives a detailed description of the cluster content.
—It is robust against noise. Our robust fitting (RF) method is specifically designed to spot and ignore noise points.
—It is space and time efficient, and thus scalable to large datasets.

To the best of our knowledge, no other clustering method meets all of the aforementioned properties. The rest of the article is organized as follows: Section 2 gives a brief survey of the large previous work. Section 3 describes our proposed framework and algorithms. Section 4 illustrates our algorithms on real and synthetic data, Section 5 proposes a more efficient variant of the algorithm, and Section 6 concludes our work.

2. SURVEY

Clustering has attracted a huge volume of interest. Recently, there have been several papers focusing on scalable clustering algorithms, such as, CURE [Guha et al. 1998], CLIQUE [Agrawal et al. 1998], BIRCH [Zhang et al. 1996], DBSCAN [Ester et al. 1996], and OPTICS [Ankerst et al. 1999]. There are also parameter-free algorithms like X-means [Pelleg and Moore 2000], and G-means [Hamerly and Elkan 2003]. However, they all suffer from one or more of the following drawbacks: They focus on spherical or Gaussian clusters, and/or are sensitive to outliers, and/or need user-defined thresholds and parameters.

Gaussian clusters. Most algorithms are geared towards Gaussian or plain spherical clusters; for example, the well-known K-means algorithm, BIRCH [Zhang et al. 1996] (which is suitable for spherical clusters), X-means [Pelleg and Moore 2000], and G-means [Hamerly and Elkan 2003]. These algorithms tend to be sensitive to outliers because they try to optimize the log-likelihood of a Gaussian, which is equivalent to the Euclidean (or Mahalanobis) distance—either way, an outlier has high impact on the clustering.

Non-Gaussian clusters. Density-based clustering methods, such as DBSCAN and OPTICS, can detect clusters of arbitrary shape and data distribution and are robust against noise. For DBSCAN the user has to select a density threshold, and for OPTICS to derive clusters from the reachability plot. K-harmonic means [Zhang et al. 2000] avoids the problem of outliers, but still needs k. Spectral clustering algorithms [Ng et al. 2001] perform K-means or similar algorithms after decomposing the $n \times n$ gram matrix of the data (typically using PCA). Clusters of arbitrary shape in the original space

correspond to Gaussian clusters in the transformed space. Here also k needs to be selected by the user. Recent interest in clustering has been on finding clusters that have non-Gaussian correlations in subspaces of the attributes [Böhm et al. 2004; Tung et al. 2005; Aggarwal and Yu 2000]. Finding correlation clusters has diverse applications ranging from spatial databases to bioinformatics.

Parameter-free methods. A smaller number of papers have focused on the subtle but important problem of choosing k, the number of clusters to shoot for. Such methods include the aforementioned X-means [Pelleg and Moore 2000] and G-means [Hamerly and Elkan 2003], which try to balance the (Gaussian) likelihood error with the model complexity. Both X-means and G-means are extensions of the K-means algorithm, which can only find Gaussian clusters and cannot handle correlation clusters and outliers. Instead, they will force correlation clusters into unnatural, Gaussian-like clusters.

In our opinion, the most intuitive criterion is based on information theory and compression. There is a family of closely related ideas, such as model-based clustering [Banfield and Raftery 1993], the gap-statistic [Tibshirani et al. 2000], the information bottleneck method [Tishby et al. 2000], which is used by Slonim and Tishby for clustering terms and documents [Slonim and Tishby 2000], and the work of Still and Bialek [2004]. Based on information theory they derive a suitable distance function for coclustering, but the number of clusters still needs to be specified in advance by the user.

There are numerous information-theoretic criterions for model selection, such as the Akaike information criterion (AIC), the Bayesian information criterion (BIC), and minimum description language (MDL) [Grünwald 2005]. Among them, MDL is the inspiration behind our VAC criterion because MDL also envisions the size of total, lossless compression as a measure of goodness. The idea behind AIC, BIC, and MDL is to penalize model complexity, in addition to deviations from the cluster centers. However, MDL is a general framework, and does not specify which distributions to shoot for (Gaussian, uniform, or Laplacian), nor how to search for a good fit. In fact, all four methods (BIC, G-means, X-means, and RIC) are near-identical for the specific setting of a noise-free mixture of Gaussians. The difference is that our RIC can also handle noise, as well as additional data distributions (uniform, etc.). We use the term *parameter-free clustering*, following the convention of all the mentioned approaches in the sense that the parameter k (number of clusters), which is particularly important and difficult to estimate, is automatically derived. This does not imply the complete absence of any parameters. In the extreme case, we could even regard the choice of distribution functions or of underlying basic algorithms as a parameter. However, our approach is (like other parameter-free methods) robust with respect to all these choices.

PCA. Principal component analysis (PCA) is a powerful method for dimensionality reduction, and is optimal under the Euclidean norm. PCA assumes a Gaussian data distribution and identifies the best hyperplane on which to project the data so that the Euclidean projection error is minimized. In other words, PCA finds global correlation structures of a dataset [Jolliffe 1986].

Table I. Table of Symbols and Acronyms

Symbol	Definition
VAC	Volume After Compression
RF	Robust Fit
CM	Cluster Merge
RIC	Robust Information-Based Clustering
n	The number of points in the dataset.
d	The dimensionality of the dataset.
$\mathcal{C}$	The clusters of a dataset, $\mathcal{C} = \{C_i \mid i = 1..k\}$.
C	A cluster of data points, $C = C_{core} \cup C_{out}$.
C_{core}	The set of core points in C.
C_{out}	The set of noise points (outliers) in C.
$\vec{x}$	A data point in S.
x_i	The ith attribute of the data point $\vec{x}$.
$\vec{\mu}$	A cluster center of cluster S.
$\vec{\mu_R}$	A robust cluster center of cluster S.
Σ (Σ_i)	The covariance matrix of points in cluster C (or C_i).
Σ_C	The conventional version of Σ (from averaging).
Σ_R	The robust version of Σ (from taking medians).
V (or V_i)	The candidate direction matrix derived from Σ (or Σ_i).
$VAC(C)$	The VAC value of points in cluster C. Small VAC value indicates that C is a good cluster.
$saveCost(C_i, C_j)$	The improvement on the VAC value of the overall clustering if C_i and C_j are merged.

3. PROPOSED METHOD

The quality of a clustering algorithm is usually sensitive to: (a) noise in the dataset, (b) algorithm parameters (e.g., number of clusters), and (c) limitations of the method used (e.g., unable to detect correlation clusters), often resulting in a unnatural partition of a dataset. Given an initial clustering of a dataset, how do we systematically adjust the clustering, overcome the influence of noise, recognize correlation patterns for cluster formation, and eventually obtain a natural clustering?

In this section, we introduce our proposed framework, RIC, for refining a clustering and discovering a most natural clustering of a dataset. In particular, we propose a novel criterion, VAC, for determining the goodness of a cluster, and propose algorithms for:

—(M1) robust estimation of the correlation structure of a cluster in the presence of noise;

—(M2) identification and separation of noise using VAC; and

—(M3) construction of natural correlation clusters by a merging procedure guided by VAC.

The proposed algorithms and VAC criterion are described in detail in the following subsections. Table I gives a list of symbols used in this work.

3.1 Goodness Criterion: VAC

The idea is to invent a compression scheme, and to declare as winner the method that minimizes the compression cost, including *everything*: the encoding for

Table II. Self-Delimiting Integer Coding

	Coding	
Number	Length	Value
1	0	1
2	00	10
3	00	11
8	0000	1000

the number of clusters k, that for the shape of each cluster (e.g., mean and covariance if it is a Gaussian cluster), and those for the cluster-id and (relative) coordinates of the data points. We assume that all coordinates are integers, since we have finite precision, anyway. In other words, we assume that our data points are on a d-dimensional grid. The resolution of the grid can be chosen arbitrarily.

The idea is best illustrated with an example. Suppose we have the dataset of Figure 1. By incorporating two distribution types, Gaussian and uniform (with a minimum bounding rectangle), in our RIC framework, the best compression (best VAC score) is achieved when the spherical cluster at the left is encoded using a Gaussian, and the linear cluster at the right is encoded by a uniform distribution.

The description of the method consists of the following parts: (a) how to encode integers; and (b) how to encode the points, once we determine that they belong in a given cluster. For (a), we propose to use self-delimiting encoding. As for (b), once we decide to assign a point to a cluster, we can store it more economically by storing its offset from the center of the cluster and using Huffman-like coding, since we know the distribution of points around the center of the cluster.

Self-delimiting encoding of integers. The idea is that small integers will require fewer bytes: We use the Elias codes, or self-delimiting codes [Chakrabarti et al. 2004], where integer i is represented using $O(\log i)$ bits. As Table II shows, we can encode the code length of the integer in unary (using $\log i$ zeros), and then the actual value by using $\log i$ more bits. Notice that the first bit of the value part is always "1", which helps us decode a string of integers without ambiguity. The system can be easily extended to handle negative numbers, as well as zero itself.

Encoding of points. Associated with each cluster C is the following information: rotatedness R (either *false* or a orthonormal rotation matrix to decorrelate the cluster), and for each attribute (regardless wheather rotated or not) the type T (Gaussian, Laplacian, uniform) and parameters of the data distribution. Once we decide that point P belongs to cluster C, we can encode the point coordinates succinctly, exploiting the fact that it belongs to the known distribution. If p is the value of the probability density function for attribute P_i, then we need $O(\log 1/p)$ bits to encode it. For a white Gaussian distribution, this is proportional to the Euclidean distance; for an arbitrary Gaussian distribution, this is proportional to the Mahalanobis distance. For a uniform distribution in, say, the minimum bounding rectangle (MBR) (lb_i, ub_i, with $0 \leq i < d$ and lb for lower

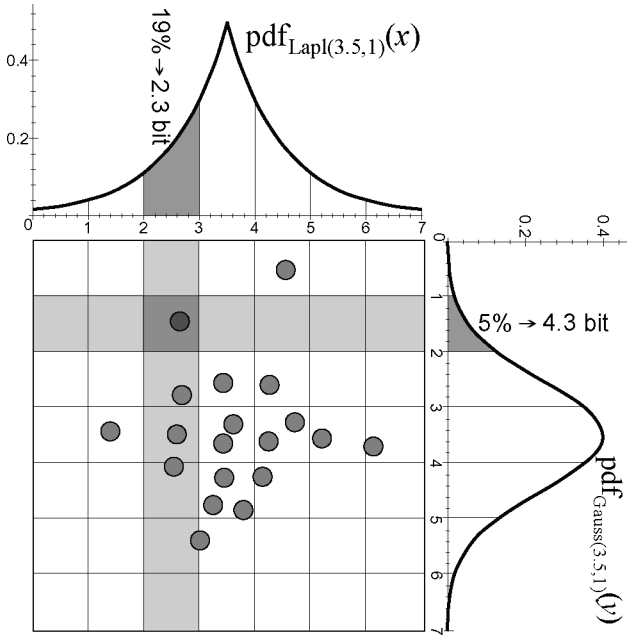


Fig. 2. Example of VAC.

bound, *ub* for upper bound, respectively), the encoding will be proportional to the area of the MBR.

The objective of this section is to develop a coding scheme for the points $\vec{x}$ of a cluster C which represents the points in a maximally compact way. For this section, we assume that all attributes of the points of the cluster have been decorrelated by PCA, and that a distribution function along with the corresponding parameters has already been selected for each attribute. In the following sections, we will describe how we identify that probability density function which gives the highest compression rate. For the example in Figure 2 we have a Laplacian distribution for the x-coordinate and a Gaussian distribution for the y-coordinate. Both distributions are assumed with $\mu = 3.5$ and $\sigma = 1$. We need to assign codewords to the coordinate values such that coordinate values with a high probability (such as $3 < x < 4$) are assigned short codewords, and those with a low probability (such as $y = 12$ to give a more extreme example) are assigned longer codewords. Provided that a coordinate is really distributed according to the assumed distribution function, Huffman codes optimize the overall compression of the dataset. Huffman codes associate to each coordinate x_i a bit-string of length $l = \log_2(1/P(x_i))$, where $P(x_i)$ is the probability of the coordinate value. Let us fix this in the following definition.

Definition 1 (*VAC of a point* $\vec{x}$). Let $\vec{x} \in \mathbb{R}^d$ be a point of a cluster C and $\overrightarrow{pdf}(\vec{x})$ be a d-dimensional vector of probability density functions which are associated to C. Each $pdf_i(x_i)$ is selected from a set of predefined probability

density functions with the corresponding parameters, that is, $PDF = \{pdf_{Gauss(\mu_i,\sigma_i)}, pdf_{uniform(lb_i,ub_i)}, pdf_{Lapl(a_i,b_i)}, \ldots\}, \mu_i, lb_i, ub_i, a_i \in \mathbb{R}, \sigma_i, b_i \in \mathbb{R}^+$. Let γ be the grid constant (distance between grid cells). The VAC_i (volume after compression) of coordinate i of point $\vec{x}$ corresponds to

$$VAC_i(\vec{x}) = \log_2 \frac{1}{pdf_i(x_i) \cdot \gamma}.$$

The VAC (volume after compression) of point $\vec{x}$ corresponds to

$$VAC(\vec{x}) = (\log_2 \frac{n}{|C|}) + \sum_{0 \le i < d} VAC_i(x).$$

In Figure 2 this is shown for the marked example point: The x-coordinate (between 2 and 3) has a probability of 19%. Thus, Huffman compression needs a total of $\log_2(1/0.19) = 2.3$ bits. The y-coordinate of this point is in a range of lower probability (5%) and needs a longer bit-string (4.3 bits). In addition to 6.6 bits for the coordinates, the Huffman-coded cluster-id is stored for each point with $\log_2(n/|C|)$ bits.

The discretization of the data space by the γ-grid enables us to interpret the obtained value as a probability (between 0 and 1) rather than a relative probability (probability density). In this respect, the VAC is more general than previous information-theoretic criteria such as BIC and AIC because it is not limited to a certain class of distribution functions, and would even allow the comparison of discrete and continuous distributions. The code lengths are always positive and correspond to the number of bits needed to compress the data using the resolution γ. It can easily be shown that the code length of each coordinate is increased by 1 bit if the number of grid cells per dimension is doubled (γ is divided by 2). This is independent of the applied probability distribution function, number of clusters, etc. However, when different datasets are clustered using different grid resolutions, the absolute values of the obtained VACs are not directly comparable. Since we only compare the VACs of different cluster structures, distribution functions, subspaces, etc., and leave the grid resolution at a constant level (high enough to distinguish the different points), the overall result of our algorithm is not sensitive to grid resolution.

Next, we address the question of which set of probability density functions has to be associated to a given cluster C. Our optimization goal is data compression, so we should, for each coordinate, select the *pdf* (and corresponding parameter setting) which minimizes the overall VAC of the cluster. It is well known that for a fixed type of *pdf* (e.g., Gaussian), the optimal parameter setting can correspond to the statistics (e.g., mean, variance, boundaries) of the dataset. Therefore, if the Gaussian *pdf* is selected for an attribute i, we use the mean and variance of the ith coordinate of the points as parameters of the *pdf*. Likewise, for the Laplacian distribution, we apply $a_i = \mu_i$ and $b_i = \sigma_i/\sqrt{2}$. For the uniform distribution, we apply the lower and upper limit of the range of the coordinate values. For the selection of the type of probability density function, we explicitly minimize the VAC of the cluster, as detailed next.

Definition 2 (*Characteristic* $\overrightarrow{pdf}(\vec{x})$ *of Cluster* C). Let C be a cluster with points $\vec{x} \in C$. Let $stat = (\mu_i, \sigma_i, lb_i, ub_i, \ldots)$ be the statistics of the data required in the set of allowed probability density functions *PDF*. Then, $\overrightarrow{pdf}$ is composed from $pdf_i \in PDF$, where

$$pdf_i = \underset{pdf_{stat} \in PDF}{argmin} \sum_{\vec{x} \in C} \log_2 \frac{1}{pdf_{stat}(x_i) \cdot \gamma}.$$

For the x-coordinate of the example in Figure 2, this means the following: First, the required statistics, namely, the mean (3.5), variance (1.0), and lower and upper limits (1.4, 6.2) of the dataset are determined. Then, VAC_x is determined for all allowed $pdf \in PDF$, that is, for $pdf_{uniform(1.4,6.2)}$, $pdf_{Gauss(3.5,1.0)}$, and $pdf_{Lapl(3.5,0.7)}$. The function yielding the lowest VAX_x is selected. Then, the same is done for VAC_y. Throughout the article we focuss on three widespread distributions of high practical relevance: Gaussian, Laplacian, and uniform. Definition 2 can easily be extended to other *pdf* functions.

Finally, we define when to use a decorrelation matrix. A decorrelation matrix is needed whenever a cluster is a correlation cluster, that is, if one (or more) attribute values of the points of the cluster depend(s) on the value of one (or more) other attributes. The decorrelation matrix can be gained from principal component analysis (PCA) of the $d \times d$ covariance matrix Σ of the points of the cluster, and corresponds to the transpose of the orthonormal matrix V gained from PCA diagonalization $V \Lambda V^T = \Sigma$. We give more details on estimating the covariance matrix in a way robust to noise in Section 3.2. Decorrelating data can greatly reduce the VAC of the cluster because, instead of having two attributes with a high variance (which incurs high coding cost for any model *pdf*) and a high correlation, we obtain two new variables without any correlation, one having variance close to zero (VAC of almost 0 bits). Intuitively, we want to use a decorrelation matrix if (and only if) the VAC improvement is considerable. To obtain a fully automatic method without user-defined limits, we use decorrelation iff the VAC savings at least compensate the effort of storing the decorrelation matrix.

Definition 3 (*Decorrelation of a Cluster*). Let C be a cluster of points $\vec{x}$ (in the original coordinate system), Σ be a covariance matrix associated to C, and V the decorrelation matrix obtained by PCA diagonalization of Σ. Let Y be the set of decorrelated points, that is, for each $\vec{y} \in Y : \vec{y} = V^T \cdot \vec{x}$. Let $\overrightarrow{pdf}(\vec{x})$ be the characteristic pdf of the original cluster and $\overrightarrow{pdf}(\vec{y})$ that of the decorrelated set Y. The decorrelation is

$$dec(C) = \begin{cases} I & \text{if } \sum_{\vec{y} \in Y} VAC(y) + d^2 f > \sum_{\vec{x} \in C} VAC(x) \\ V & \text{otherwise.} \end{cases}$$

The information on which of the two cases is true is coded by 1 bit. The matrix V is coded using $d \times d$ floating values using f bits. The identity matrix needs

no coding (0 bits).

$$VAC(dec(C)) = \begin{cases} 1 \text{ if } \sum_{\vec{y} \in Y} VAC(y) + d^2 f > \sum_{\vec{x} \in C} VAC(x) \\ d^2 \cdot f + 1 \text{ otherwise} \end{cases}$$

The following definition puts these things together.

Definition 4 (*Cluster Model*). The cluster model of a cluster C is composed from the decorrelation $dec(C)$ and the characteristic $\overrightarrow{pdf}(\vec{y})$, where $\vec{y} = dec(C) \cdot \vec{x}$ for every point $\vec{x} \in C$. The volume after compression of the cluster $VAC(C)$ corresponds to

$$VAC(C) = VAC(dec(C)) + \sum_{\vec{x} \in C} VAC(dec(C) \cdot \vec{x}).$$

3.2 Robust Fitting (RF)

We consider the combination of the cluster's subspace and the characteristic probability distribution as the *cluster model*. A data point in a (tentative) cluster could be either a *core point* or an *outlier*, where core points are defined as points in the cluster's subspace which follow the characteristic probability distribution of the cluster model, while outliers are points that do not follow the distribution specified by the cluster model. We will also call the outliers *noise (points)*.

Having outliers is one factor that prevents conventional clustering methods from finding the right cluster model (using, e.g., PCA). If the cluster model is known, filtering outliers is relatively easy; just remove those points which fit the worst, according to the cluster model. Likewise, determining the model when clusters are already purified from outliers is equally simple. What makes the problem difficult and interesting is that we have to filter outliers without knowledge of the cluster model and vice versa.

Partitioning clustering algorithms, such as those based on K-means or K-medoids, typically produce clusters that are mixed with noise and core points. The quality of these clusters is hurt by the existence of noise, which leads to a biased estimation of the cluster model.

We propose an algorithm for purifying a cluster in which, after the processing, noise points are separated from their original cluster and form a cluster of their own. We start with a short overview of our purification method before going into the details. The procedure starts with getting as input a set of clusters $\mathcal{C}=\{C_1, \ldots, C_k\}$ by an arbitrary clustering method. Each cluster C_i is purified one-by-one: First, the algorithm estimates an orthonormal matrix called the *decorrelation matrix (V)* to define the subspace of cluster C_i. A decorrelation matrix defines a similarity measure (an ellipsoid) which can be used to determine the boundary that separates core points and outliers. Our procedure will pick the boundary which corresponds to the lowest overall VAC value of all points in cluster C_i. The noise points are then removed from that cluster and stored in a new one. Next, we elaborate on the steps for purifying a cluster of points.

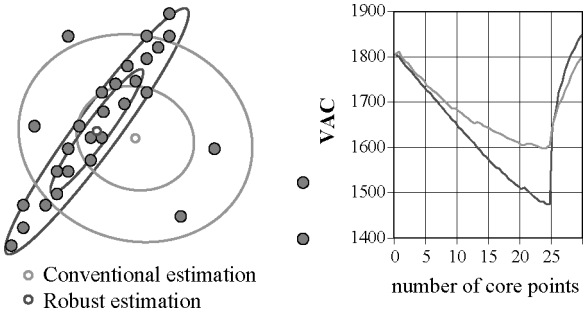


Fig. 3. Conventional and robust estimation.

3.2.1 *Robust Estimation of the Decorrelation Matrix.* The decorrelation matrix of a cluster C_i contains the vectors that define (span) the space in which points in cluster C_i reside. By diagonalizing the covariance matrix Σ of these points using PCA ($\Sigma = V \Lambda V^T$), we obtain an orthonormal Eigenvector matrix V, which we defined as the decorrelation matrix. The matrices V and Λ have the following properties: The decorrelation matrix V spans the space of points in C, and all Eigenvalues in the diagonal matrix Λ are positive. To measure the distance between two points $\vec{x}$ and $\vec{y}$, taking into account the structure of the cluster, we use the Mahalanobis distance defined by Λ and V.

$$d_{\Sigma_C}(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^T \cdot V \cdot \Lambda^{-1} \cdot V^T \cdot (\vec{x} - \vec{y})$$

Given a cluster of points C with center $\vec{\mu}$, the conventional way to estimate the covariance matrix Σ is by computing a matrix Σ_C from points $\vec{x} \in C$ by the averaging

$$\Sigma_C = 1/|C| \sum_{\vec{x} \in C} (\vec{x} - \vec{\mu}) \cdot (\vec{x} - \vec{\mu})^T,$$

where $(\vec{x} - \vec{\mu}) \cdot (\vec{x} - \vec{\mu})^T$ is the outer vector product of the centered data. In other words, the (i, j)-entry of the matrix Σ_C, $(\Sigma_C)_{i,j}$, is the covariance between the ith and jth attributes, which is the product of the attribute values $(x_i - \mu_i) \cdot (x_j - \mu_j)$, averaged over all data points $\vec{x} \in C$. Moreover, Σ_C is a $d \times d$ matrix, where d is the dimension of the data space.

The two main problems of this computation when confronted with clusters containing outliers are that: (1) the centering step is very sensitive to outliers, namely, in that outliers may heavily move the determined center away from the center of the core points, and (2) the covariances are heavily affected from wrongly centered data and from the outliers as well. Even a small number of outliers may thus completely change the complete decorrelation matrix. This effect can be seen in Figure 3 where the center has been wrongly estimated using conventional estimation. In addition, the ellipsoid which shows the estimated "data spread" corresponding to the covariance matrix has a completely wrong direction that is not followed by the core points of the clusters.

To improve the robustness of the estimation, we apply an averaging technique which is much more outlier robust than the arithmetic means: the coordinate-wise median. To center the data, we determine the median of each

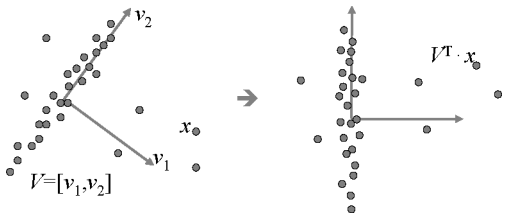


Fig. 4. The decorrelation matrix.

attribute independently. The result is a dataset where the origin is close to the center of core points of the cluster ($\vec{\mu_R}$), rather than the center of all points ($\vec{\mu}$).

A similar approach is applied for the covariance matrix: Here, each entry of the robust covariance matrix $(\Sigma_R)_{i,j}$ is formed by the median of $(x_i - \mu_{R_i}) \cdot (x_j - \mu_{R_j})$ over all points $\vec{x}$ of the cluster. The matrix Σ_R reflects more faithfully the covariances of core points, compared to the covariance matrix obtained by the arithmetic means.

The arithmetic-mean covariance matrix Σ_C has the *diagonal dominance* property, where the each diagonal element $\Sigma_{i,i}$ is greater than the sum of other elements of the row $\Sigma_{*,i}$. The direct consequence is that all Eigenvalues in the corresponding diagonal matrix Λ are positive, which is essential for the definition of $d_\Sigma(\vec{x}, \vec{y})$.

However, the robust covariance matrix Σ_R might not have the diagonal dominance property. If Σ_R is not diagonally dominant, we can safely add a matrix $\phi \cdot I$ to it without affecting the decorrelation matrix. The value ϕ can be chosen as the maximum difference of all column sums and the corresponding diagonal element (plus some small value, say 10%).

$$\phi = 1.1 \cdot \max_{0 \le i < d} \left\{ \left(\sum_{0 \le j < d, i \ne j} (\Sigma_C)_{i,j} \right) - (\Sigma_C)_{i,i} \right\}$$

It can easily be seen that adding the matrix ϕI does only affect the Eigenvalues and not the Eigenvectors: If $\Sigma = V \Lambda V^T$, then $\Sigma + \phi I = V \Lambda V^T + \phi I$. Since V is orthonormal, ϕI can also be written as $V \phi I V^T$, and due to the distributive law we have $\Sigma + \phi I = V(\Lambda + \phi I)V^T$, that is, each Eigenvalue is increased by ϕ and matrix V is unaffected by this operation.

Using our robust estimation technique, the center in Figure 3 is correctly positioned and the ellipsoid which represents the covariance matrix follows the distribution of core points. The safe decorrelation matrix V (compare with Figure 4), which has been generated from the safely estimated covariance matrix, is composed from Eigenvectors which indicate the directions of maximum variance of the core of the cluster. When transforming the data by multiplication of V^T, we remove the correlations of the attributes. Note that we do not decide about a projection into a lower-dimensional space at this stage, that is, there is no information loss.

3.2.2 *Partitioning Points into Core and Noise.* The first step of purifying a cluster of points is to identify the proper decorrelation matrix. We generate several estimates (called *candidates*) of the covariance matrix, using various

estimation methods, and pick the one with the best overall VAC value. In our experiments, the candidates include the matrix Σ_C from the conventional method using arithmetic average, and matrix Σ_R from the robust method described earlier. We also determine a conventional and a robust candidate, matrices $\Sigma_{C,50}$ and $\Sigma_{R,50}$, respectively, by considering only a certain percentage (e.g., 50%) of points in the cluster being closest to the robustly estimated center $\vec{\mu_R}$. In addition, we always have the identity matrix I as one candidate decorrelation matrix. Among these matrices, our algorithm selects the matrix giving the best (lowest) overall VAC. For our example in Figure 3, the diagram at the right shows that the lowest VAC value of 1480 is reached for robust estimation in contrast to 1600 for conventional estimation.

The next step is to detect noise points in the cluster. By now, we have computed the robust center $\vec{\mu_R}$, and have chosen a candidate covariance matrix which we call $\Sigma*$ (the corresponding decorrelation matrix is $V*$). The goal is to partition the set of points in cluster C into two new sets: C_{core} (for core points) and C_{out} (outliers). First, our method orders the points of C according to the Mahalanobis distance defined by the candidate covariance matrix $\Sigma*$. Initially, we define all points to be outliers ($C_{out} = C, C_{core} = \{\}$). Then, we iteratively remove points $\vec{x}$ from C_{out} (according to Mahalonobis sort order starting with the point closest to the center) and insert them into C_{core}, and compute the coding costs before and after moving the point $\vec{x}$.

At each iteration, the point $\vec{x}$ being moved from C_{out} to C_{core} is first projected to the space defined by the selected candidate decorrelation matrix $V*$. Then, the coding cost of the new configuration ($C_{core} \cup \{\vec{x}\}, C_{out} - \{\vec{x}\}$) is determined as the cost where each of the coordinates is modeled using that distribution function which gives least coding costs. Outlier points are always coded using uniform distribution. So, each of these configurations corresponds to one given radius of the ellipsoid partitioning the set into core and noise objects. The partition which has the least overall cost in this algorithm is finally returned (refer to Figure 3 where at the minimum of 1480, we have 24 objects in the core set and 6 in the noise set). The diagram in Figure 3 depicts the VAC value (v-axis) of the different configuration (C_{core}, C_{out}) at each iteration (x-axis). Figure 3 shows two VAC-value curves: one for the conventional candidate decorrelation matrix (V_C) and the other for the robust estimation (V_R). At the beginning, all points are regarded as noise points, yielding a VAC value of approximately 1800 for both candidate matrices. As more and more points are moved from C_{out} to the set of core points C_{core}, the VAC value improves (decreases). For the robust decorrelation matrix (V_R), the VAC value reaches the minimum of 1480 when there are 24 core points. After this, the VAC value increases again to approximately 1800.

3.3 Cluster Merging (CM)

Our RIC framework is designed to refine the result of any clustering algorithm (e.g., K-means). Due to imperfection of the clusters given by an algorithm, our cluster purifying algorithm may lead to redundant clusters containing noise objects that fit well to other neighboring noise clusters. In this section we describe

our proposed cluster merging procedure in more detail, to correct the wrong cluster assignments caused by the original clustering algorithm.

For example, the K-means clustering algorithm tends to partition data incorrectly when the true clusters are noncompact. These clusters are often split up into several parts by K-means. A typical, inappropriate partitioning is shown in Figure 6(a). Our algorithm corrects the wrong partitions by merging clusters that share common characteristics, taking into account the subspace orientation and data distribution. We use the proposed VAC value to evaluate how well two clusters fit together. The idea is to check whether the merging of a pair of clusters could decrease the corresponding VAC values. Mathematically, let $VAC(C)$ be the VAC value for a cluster C. We also define $savedCost(C_i, C_j)$ of a cluster pair (C_i, C_j) as

$$savedCost(C_i, C_j) = VAC(C_i) + VAC(C_j) - VAC(C_i \cup C_j).$$

If $savedCost(C_i, C_j) > 0$, then we consider the cluster pair (C_i, C_j) a potential pair for merging. Our proposed merging process is an iterative procedure. At each iteration, our algorithm merges those two clusters which have the maximum $savedCost(.,.)$ value, resulting in a greedy search toward a clustering that has the minimum overall cost. To deter this greedy algorithm from getting stuck in a local minimum, we do not stop immediately, even when no savings of $savedCost(.,.)$ value can be achieved by merging pairs of clusters. In other words, we do not stop when $savedCost(.,.) \leq 0$. Instead, the algorithm continues for another t iterations, continuing to merge cluster pairs (C_i, C_j) with the maximum $savedCost(C_i, C_j)$ value, even though now the $savedCost(C_i, C_j)$ value is negative and merging C_i and C_j will increase the VAC value of the overall dataset. Whenever a new minimum is reached the counter is reset to zero. Pseudocode for the RIC algorithm is given in Figure 5.

Complexity analysis. The RF algorithm considers $(n-1)$ partitions of the dataset into core and noise points. For each partion the VAC is determined, which requires $O(n \cdot d^3)$ time due to PCA, so the overall complexity is $O(n^2 \cdot d^3)$. The CM algorithm is quadratic in the number of initial clusters C_{init}. In each merging step, the VAC of the new cluster needs to be determined, which leads to an overall complexity of $O(C_{init} \cdot n \cdot d^3)$. The overall complexity of standard RIC is quadratic in n and cubic in d. In Section 5 a more efficient solution is proposed.

4. EXPERIMENTS

4.1 Results on Synthetic Data

Especially widespread K-means and K-medoid clustering methods often fail to separate clusters from noise, and therefore produce results where the actual clusters are contaminated by noise points. Figure 6(a) shows the result of K-means with $k = 8$ on a synthetic 2D dataset consisting of 4751 data objects. Two of the resulting clusters contain many noise objects, among them the 1D correlation cluster. In Figure 6(b) the result of the cluster purifying algorithm is depicted. Five of the eight initial clusters have been split up into

Data structure clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$
Each cluster C_i, has two members:
C_i.points: points in cluster C_i.
C_i.VAC: the VAC value of cluster C_i.

algorithm refined clusters $\mathcal{R}$ = RIC (initial clusters $\mathcal{C}$)
clusters $\mathcal{P}$:= RobustFitting($\mathcal{C}$);
clusters $\mathcal{R}$:= ClusterMerging($\mathcal{P}$);
return refined clusters $\mathcal{R}$;

algorithm clusters $\mathcal{P}$ = RobustFitting(initial clusters $\mathcal{C}$)
// Purifying clusters from noise.
Initialize the output clusters $\mathcal{P} = \{\}$. **for each** cluster $C \in \mathcal{C}$
Estimate the direction matrix;
Search for the best split of C into C_{core} (core objects) and C_{out} (noise objects), according to the minimal VAC value;
Initialize the VAC values of C_{core} and C_{out}.
$\mathcal{P} = \mathcal{P} \cup \{C_{core}, C_{out}\}$;
return purified clusters $\mathcal{P}$;

algorithm clusters $\mathcal{C}$ = ClusterMerging(clusters $\mathcal{C}$, int t)
//Merging purified clusters.
while $|C| > 1$ **and** $savedCost > 0$
Find the best pair of clusters to merge:
$[(C*_1, C*_2)$, mergedVAC($C*_1$,$C*_2$)]= findMergePair(clusters $\mathcal{C}$);
Merge $C*_1$ and $C*_2$ as $C_{new} = \{C*_1 \cup C*_2\}$:
$\mathcal{C} = \mathcal{C} - \{C*_1, C*_2\} \cup \{C_{new}\}$.
Set VAC(C_{new}) := mergedVAC($C*_1$,$C*_2$);
end while

while $|C| > 1$ **and** $counter < t$
//Improved search: Getting out of local minimum
Find the best pair of clusters to merge:
$[(C*_1, C*_2)$, mergedVAC($C*_1$,$C*_2$)]= findMergePair(clusters $\mathcal{C}$);
$counter++$;
Merge $C*_1$ and $C*_2$ as $C_{new} = \{C*_1 \cup C*_2\}$
Set VAC(C_{new}) := mergedVAC($C*_1$,$C*_2$);
end while
return the clustering $\mathcal{C}$ with the minimum overall VAC value, found during the t iterations;

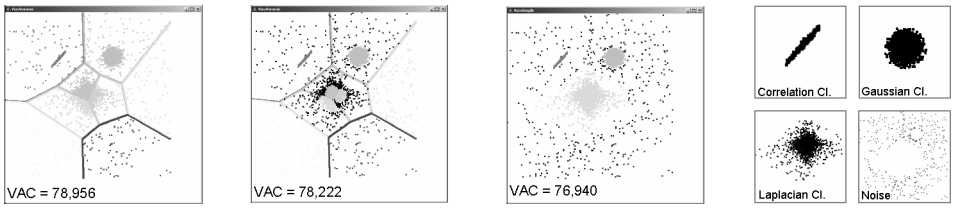
subroutine $[(C*_1, C*_2)$, mergedCost($C*_1$,$C*_2$)]= findMergePair(clusters $\mathcal{C}$)
// Find cluster pair with the best (maximal savedCost).
for all cluster pairs $(C_i, C_j) \in \mathcal{C} \times \mathcal{C}$
mergedVAC(C_i,C_j) := VAC($C_i \cup C_j$);
savedCost(C_i,C_j):=(VAC(C_i)+VAC(C_j))-mergedVAC(C_i,C_j);
find the cluster pair to merge: $(C*_1, C*_2)$=$argmax_{(C_i,C_j)}$savedCost(C_i, C_j);
return The cluster pair $(C*_1, C*_2)$, and their mergedCost($C*_1$,$C*_2$);

Fig. 5. RIC algorithm.

clusters containing noise objects and clusters with core points. Three of the initial clusters contain only noise objects. No objects need to be filtered out, so these partitions remain unchanged. The purifying algorithm reduces the overall VAC from 78,956 to 78,222.

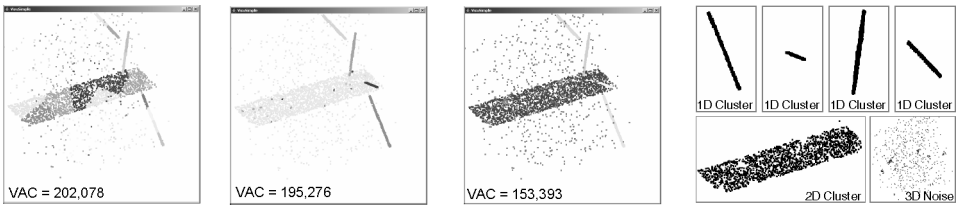
As a building block we provide fully automatic noise filtering and outlier detection. Our approach is model-based and supports both subspace and correlation clusters as well as various data distributions. It provides a natural cut-off point for the property of being an outlier based on the coding cost.

After the initial clusters have been purified, our algorithm merges together those with common characteristics such as common subspace orientation or



(a) K-means (b) after purifying (c) RIC result (d) detailed view

Fig. 6. 2D synthetic data.



(a) K-means (b) DBSCAN (c) RIC result (d) detailed view

Fig. 7. 3D synthetic data.

data distribution. In the example depicted in Figure 6(a), the cluster in the center has been split up into three parts by K-means. This inappropriate partitioning is corrected by the cluster merging algorithm (compare with Figure 6(c)). Also, the noise clusters generated by the previously applied cluster purifying algorithm are now merged. The resulting clustering in our example consists of four clusters. The cluster merging algorithm drastically reduces the VAC score by removing redundant clusters.

As a particular advantage over conventional clustering, RIC provides information on the data distribution of the coordinates. In our example, the x-coordinate of the correlation cluster (top left in Figure 6(d)) is uniformly distributed, the y-coordinate Gaussian. Both coordinates of the top right cluster follow a Gaussian distribution. Both coordinates of the bottom left cluster are Laplacian, and both of the bottom right (representing noise objects) are uniformly distributed.

We demonstrate the performance of the cluster filtering and merging algorithm on a 3D synthetic data containing 7500 data objects (cf. Figure 7). This dataset consists of one plane (2D correlation cluster, 2000 objects) and three lines (1D correlation clusters, two with 2000 objects each, one with 1000), and 500 noise objects. Note that one of the lines is embedded in the plane. Figure 7(a) shows the clustering result of K-means with $k = 20$. The correlation clusters are split up in several parts and the noise objects distributed among all clusters. This initial clustering obtains a VAC score of 202,078. After applying the cluster purifying and merging algorithm, we obtain a much better clustering result with VAC 153,393. Further, 98.6% of the noise objects are correctly assigned to the noise cluster. The plane is 94.6% pure and the lines, even the one embedded in the plane, are from 99.5% to 100% pure.

Table III. Clusters Found by RIC

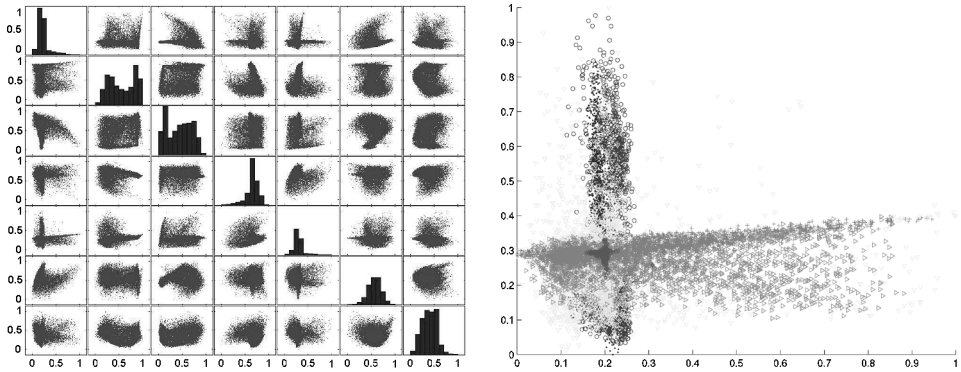
Method	c-id	Control	PKU	VAC	IMP
RIC+K-means	1	0	275		
	2	337	31	74,298	31
K-means $k = 2$	1	0	222		
	2	337	84	75, 497	84
RIC+spectral	1	2	282		
	2	335	24	72,131	26
spectral $k = 2$	1	2	224		
	2	335	82	75,922	84

The DBSCAN algorithm ($MinPts = 4, \epsilon = 0.1$) correctly detects the lines, but fails to separate the plane from the noise objects, and creates many small clusters in dense areas of the plane (compare with Figure 7(b)). There are 34 initial clusters in total. This result has a VAC score of 195,276. After the purifying and merging algorithm we obtain a VAC of 155,412 and a very similar result, as depicted in Figure 7(c). This demonstrates that the RIC framework can be applied with various partitioning clustering methods. Since the dataset has been artificially generated, we can determine the VAC for the ideal clustering (exactly corresponding to the generated clusters): The VAC of the ideal clustering (151, 637) is almost reached by RIC after K-means as well as RIC after DBSCAN.

The greedy fashion optimization process is efficient. We implemented the RIC algorithm in Java. Runtimes for synthetic datasets are 147 s for the 2D dataset and 567 s for the 3D dataset on a PC with 3 GHz CPU and 1 GB RAM.

4.2 Performance on Real Data

4.2.1 *Metabolic Data.* We evaluate the RIC framework using a high-dimensional metabolic dataset. This 14D dataset (643 instances) was produced by modern screening methodologies and represents 306 cases of PKU, a metabolic disorder, and 337 objects from a healthy control group. As initial clusterings, we used spectral clustering (with $d = 12$ dimensions), and K-means; in both cases we used $k = 6$ initial clusters. To evaluate class purity of the clusterings, we report IMP, the count of *impurities*, defined as the count of minority points in each cluster. The initial clusterings have an IMP of 31 and a VAC of 77,822 for K-means and an IMP of 26 and a VAC of 78,184 for spectral clustering, respectively. Table III shows the same quantities and the clustering results after we apply RIC. We obtain the best result by applying RIC after spectral clustering. Two out of 14 features from cluster 1 hosting class PKU are uniformly distributed; the remaining features follow the Laplacian distribution. All features of cluster 2 containing the control group follow the Laplacian distribution. Notice that in all cases, RIC achieved everything we wanted: (a) It found the correct number of clusters, and (b) it achieved better compression (lower VAC score, as expected). For comparison, we also show the results of K-means and spectral clustering after setting $k = 2$ (which gives an unfair advantage over RIC). Even so, notice that RIC achieves both lower VAC scores as well as better impurity count IMP. Using $k = 2$, both K-means and



(a) (b)

Fig. 8. (a) Visualizing the distribution of 7D retinal image tiles. Each subfigure shows the distribution of two dimensions. The dataset contains non-Gaussian clusters; (b) the 13 clusters found by RIC.

spectral clustering assign many instances of class PKU to the cluster of the control group.

4.2.2 *Cat Retina Images.* The data we consider here consists of image blocks in retinal images from the UCSB BioImage (`http://bioimage.ucsb.edu/`) database. The blocks are taken from 219 images of the retina under 9 different conditions (healthy, diseased, etc.). Each image is of size 512×768. We take nonoverlapped pixel blocks (called *tiles*) of size 64×64 from each image, and collect in 96 tiles per image, or 21,024 tiles in total. Each tile is represented as a vector of 7 features, exactly as suggested in Bhattacharya et al. [2005]. Figure 8(a) depicts the distribution of image tiles. The distribution is viewed from all possible pairs of dimensions; the (i, j)-subfigure plots the ith dimension versus the jth. The histograms at the diagonal subfigures depict the distribution of values in each dimension. The retina image tiles clearly have a non-Gaussian distribution with correlation clusters. Some views show strong correlation patterns, for example, the view of the 1st and 5th dimensions (the subfigure at the first row and fifth column). In the following discussion, we will focus on the view of the 1st and 5th dimensions, and show that our RIC framework is able to find the non-Gaussian correlation clusters in this dataset.

Moreover, most of the coordinates in the detected clusters clearly show a super-Gaussian distribution which is reported as Laplacian by RIC. Let us note that our framework is extensible and can incorporate every data distribution that has a *pdf*. Figure 8(b) shows the RIC clustering result on the retinal tiles, where points of a cluster are plotted with an unique symbol. In total, RIC produces 13 clusters for this dataset. We plot each cluster separately in different figures for better visualization of the clustering result.

Some plots of individual clusters are shown in Figures 9(a)–(f).

It can be easily seen that the proposed RIC method successfully finds the correlation clusters in this dataset, and, unlike other methods like K-means, will neither overcluster nor undercluster the dataset.

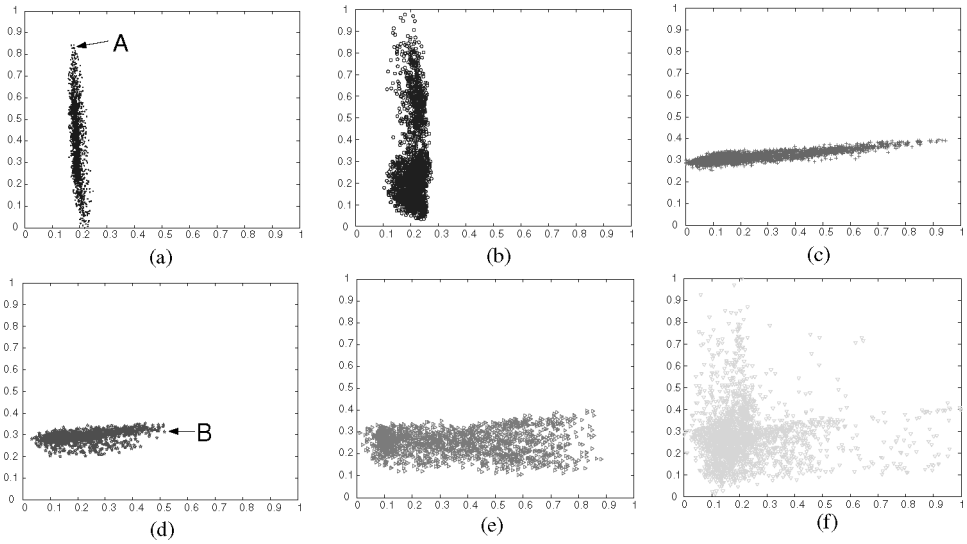


Fig. 9. Example clusters on retinal image tiles found by RIC.

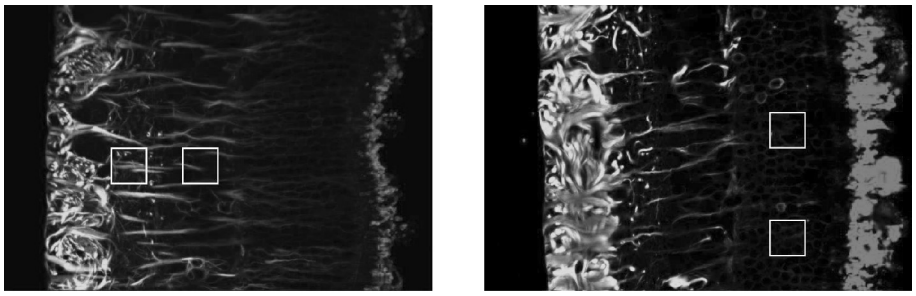


Fig. 10. The white boxes in the two retinal images indicate example tiles in selected clusters. Left (a) tiles at position **A** of cluster of Figure 9(a); right (b) tiles at position **B** of cluster of Figure 9(d).

The question is: Is there any biological meaning to the clusters derived by RIC? "The answer is yes." Tiles from cluster (A) (see Figure 9(a)) are shown in Figure 10(a), and tend to correspond to the so-called "Müller cells." Similarly, tiles from cluster (B) (see Figure 9(d)) are shown in Figure 10(b), and tend to correspond to the so-called "rod photoreceptors."

Specifically, Figure 10 shows the layers of cells of a cat's retina. The lighter colors in the image indicate the distribution of two proteins (rod opsin and GFAP). In Figure 10(a), the white boxes highlight two tiles at position **A** of the cluster shown in Figure 9(a). The image shows the situation of a layer-detached retina being treated with oxygen exposition. Highlighted tiles are Müller cells with protein GFAP propagated from the inner layer of the retina.

In Figure 10(b), the white boxes highlight two tiles at position **B** of the cluster shown at Figure 9(d). The image shows the case of a retina which has suffered

layer detachment for three months. The tiles highlighted are rod photoreceptors, with the protein rod opsin redistributed into the cell bodies; this is typical for detached retinas.

The point is that our clustering method, without *any* domain knowledge, manages to derive groups of tiles that do have biological meaning (Müller cells and rod photoreceptors, respectively).

5. EXTENSIONS

In this section, we propose several extensions to the RIC framework. In particular, we propose more effective and efficient methods for filtering non-Gaussian clusters. In addition, we present an efficient top-down clustering method which has been designed to optimize the VAC criterion. An extensive experimental evaluation demonstrates that this algorithm is a good choice as stand-alone clustering method, that is, no preclustering by another clustering algorithm is required.

5.1 Purifying Clusters from Noise

As discussed in Section 3.2, it is an essential step of our algorithm to purify the initial clusters from noise. The model and subspace orientation of the clusters can be safely determined by the robust fitting method (refer to Section 3.2.1). Subsequently, the points of a cluster can be separated into core points and noise points (refer to Section 3.2.2).

5.1.1 *FilterCost: Filtering by Coding Cost.* The algorithm for partitioning the points of a cluster into core and noise points, described in Section 3.2.2, works well for clusters with a data distribution that is at least approximately Gaussian. Recall that the main idea of this algorithm, subsequently called *filterDist*, is to sort the points with respect to their distance from the robustly estimated cluster center. The cluster is rotated according to its robustly estimated decorrelation matrix if the saved cost due to this rotation pays off the coding cost of the decorrelation matrix. In this case, the Mahalonobis distance is used for sorting the points with respect to their distance from the cluster center. The algorithm then determines the coding cost for each possible partitioning into core and noise points according to this sorting order. For clusters that have a data distribution clearly differing from Gaussian, this method is not optimal because the sorting order does not correspond well to the data distribution.

However, considering a point $\vec{x}$ and a cluster C, the VAC of $\vec{x}$ as specified in Definition 1 determines its correct position in the sorting order with respect to C. The VAC of $\vec{x}$ needs to be encoded using the characteristic $\overrightarrow{pdf}(\vec{x})$ of cluster C (refer to Definition 2). Based on this observation, we can easily modify the filterDist algorithm to achieve better filtering results for non-Gaussian, for example, Laplacian, clusters. Again, we code the noise points using the *pdf* for uniform distribution. The filtering algorithm starts with all points being outliers, namely ($C_{out} = C, C_{core} = \{\}$). As described in Section 3.2.2, the algorithm removes at each iteration one point from the set of outliers and adds it to the set of cluster points. The coding cost for the new configuration is determined

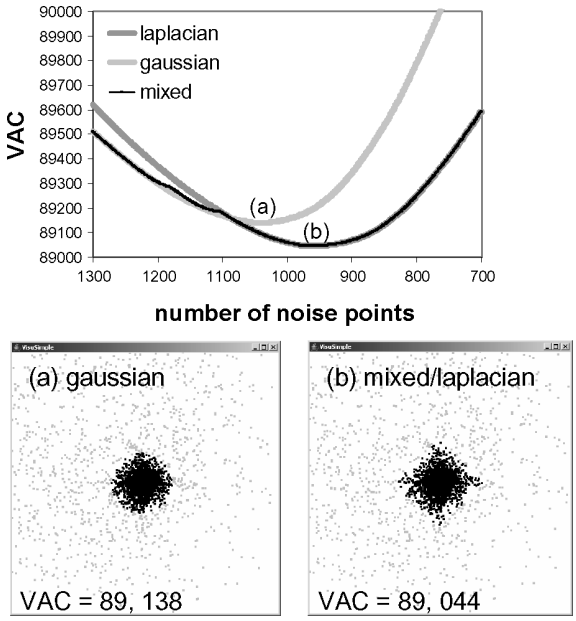


Fig. 11. Mixed filtering.

and the cluster points are coded according $\overrightarrow{pdf}(\vec{x})$. The main difference is that the sorting order applied now is based on the coding costs of the points with respect to the characteristic $\overrightarrow{pdf}(\vec{x})$ of the corresponding cluster.

Since sorting order now depends on coding cost, the question arises as to whether it is necessary to run the filtering algorithm more than once. In principle, for each coordinate, one of the allowed distribution functions can be independently selected, which results in a total number of $|PDF|^d$ different sort orders for each cluster C. However, the problem of an exponential number of runs of the filtering algorithm can be completely avoided by the following modification of our method, which needs only *one* run instead of $|PDF|^d$ different runs. Our algorithm still starts with a setting where all points are considered outliers. However, our points are not sorted according to a fixed order. Instead, in each step, the point which causes the highest cost gain is selected and inserted into the set of core points. Then the cluster model is updated. This means that the characteristic probability density function ($\vec{pdf}(\vec{x})$) for the current set of cluster points and the corresponding parameters (μ_i, σ_i) are redetermined and will be used in the next step of our loop. Thus, the sorting order is allowed to change during the run of the filtering algorithm.

As an example, Figure 11 displays the VAC of a 2D example consisting of a Laplacian cluster of 2000 objects which is embedded in an environment of 1000 uniformly distributed noise objects. The diagram displays a varying association of objects to noise and core, ranging from 1300 to 700 noise objects (in descending order, analogously to the considered order of the algorithm). For comparison we show the cost for a fixed sort order under the assumptions that both coordinates are Laplacian and that both are Gaussian, respectively. The

interesting observation here is that our new method of an updated sort order (depicted in the thin, black line and denoted "mixed") almost always adopts the better of the two cost curves with fixed *pdf* association (our algorithm switches from a Gaussian to a Laplacian model at approximately 1100 noise objects). This is particularly true at the overall optimum of the association, which correctly partitions the dataset into 964 noise points and 2036 cluster points with a VAC of 89,044 (position (b)). Note that some of the points which have been generated as noise points are indeed located inside the cluster and can therefore not be distinguished from core points, which explains the fact that 36 more objects are considered core objects. We can also learn from this diagram that (not surprisingly) the Gaussian coding is not so appropriate (VAC = 89,138, position (a)) for this dataset, as it reaches a local optimum for approximately 1100 noise points.

Let us note that the result of the filterDist algorithm is similar to Gaussian filtering because the Euclidean (weighted Euclidean and Mahalonobis) distance is equivalent to sorting based on Gaussian coding cost. Compared to distance-based filtering, the cost-based filtering algorithm, henceforth called *filterCost*, achieves better results on non-Gaussian clusters. However, the runtime is quadratic in the size $|C|$ the cluster (due to selection of the maximum cost gain in each iteration of the loop). Therefore, we propose a very efficient solution to the cluster filtering problem in the next section.

5.1.2 *FilterOpt: An Optimization-Based Filtering Algorithm.* Although the example depicted in Figure 11 is rather difficult (one-third of all points are noise points), the VAC curve of the filter cost algorithm shows a distinct cost minimum indicating the correct partitioning. Provided that the dataset to be filtered only contains core points of the cluster and noise points, we observe a VAC curve with a distinct global cost minimum. To find this minimum, it is not required to examine all possible $n-1$ segmentations into core and noise points. For this task, a much more efficient iterative algorithm optimizing the VAC can be employed.

The main idea of this algorithm is to use a K-Means-style algorithm which terminates after a few iterations to separate core from noise points. In each partition from the initial clustering to be filtered, we run this algorithm for $k = 2$ clusters representing the core points and noise points, respectively. However, the K-Means paradigm requires major modifications for our purposes. First, as in the filterCost algorithm, we use the coding costs instead of a metric distance function when assigning points to clusters. In addition, the cluster which should finally contain the core points and that which should end up containing the noise points (in the following called the *noise cluster*) cannot be handled equally during the run of the algorithm. Otherwise, both core and noise points are distributed over both clusters because the 2-Means algorithm aims at equally optimizing the coding costs of both clusters. We avoid this effect by:

(1) fixing both cluster centers at the robustly estimated center of the core points; and
(2) selecting an appropriate coding scheme for the noise cluster.

```
algorithm FilterOpt (cluster C, pdf)
// Purifying clusters from noise.
  create clusters C_core C_noise;
  set the centers of C_core C_noise to median(C);
  set the variances of C_core as the variances of C;
  set the variance of C_noise to a large value;
  use pdf for C_core and gaussian for C_noise;
  init();
  do until convergence
    assign all objects o ∈ C to C_core or C_noise depending on minimum VAC;
     update center of C_core and C_noise and as median(C_core);
    update variance of C_core
  return C_core and C_noise
```

Fig. 12. FilterOpt algorithm.

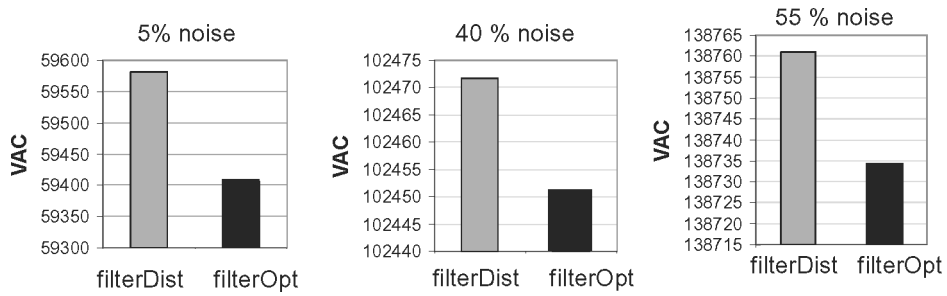


Fig. 13. Effectiveness of the FilterOpt algorithm.

The first aspect is motivated as follows: In a K-means-like approach, two separate, nonequal centers for core and noise clusters always define a border such that all objects at one side of the border are associated to the core cluster and all objects on the other to the noise cluster. In subsequent iterations, typically the centers become even more separated. In traditional K-means with Euclidean distance, this border is a straight line (or hyperplane for data spaces of dimensionality greater than 2), resulting in a Voronoi diagram for the complete clustering. For ellipsoid distances and more complex distance measures (including our cost-based assignment) the border is no longer a straight line, but a higher-degree function; nonetheless, the border does still exist. In contrast, fixing both centers in one common point allows the separation of two nested datasets which correspond to different data distributions (noise and core). The second aspect guides the algorithm to the desired result. We model the noise cluster as a Gaussian distribution with high variance. The pseudocode of the algorithm is presented in Figure 12. The algorithm can be applied in rotated and unrotated space. For initialization, we selected to assign the 50% of objects having higher coding costs to noise. We now compare this algorithm, called *filterOpt*, to filterDist and filterCost in terms of effectiveness and efficiency.

Effectiveness. On the example depicted in Figure 11, filterOpt achieves a very similar result as filterCost, with a VAC of 89,051. Figure 13 compares the

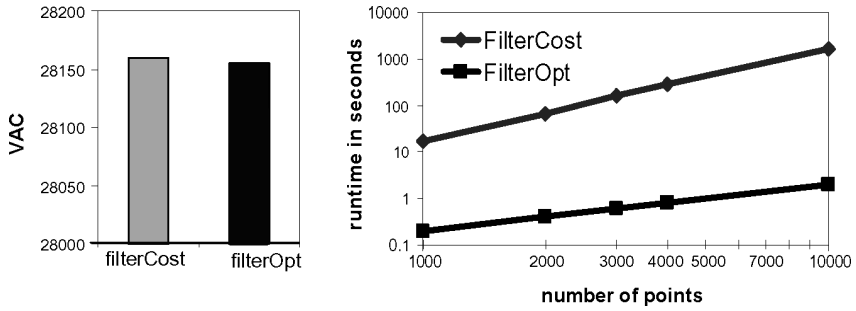


Fig. 14. Efficiency and effectiveness of FilterOpt and FilterCost.

result of filterOpt to the result of filterDist on a 2D dataset containing a Laplacian cluster and various amounts of noise. In all settings, filterOpt achieves a lower VAC than filterDist.

Efficiency. The filterOpt algorithm is very efficient with a runtime complexity $O(iter \cdot n)$, where *iter* is the number of iterations. Usually, the algorithm terminates after a few iterations; for example, it takes 7 iterations to filter a dataset similar to the one depicted in Figure 11 with 1000 points, which takes 328 milliseconds. The filterCost algorithm needs 17 seconds for the same dataset. A data set with 10,000 points can be processed by filterOpt in 2 seconds, whereas the filterCost algorithm needs 28 minutes. Figure 14 shows the runtime in seconds for various sizes of the data set.

5.2 A Top-down Clustering Algorithm

In this section, we propose a top-down clustering algorithm which uses the filterOpt algorithm as a building block. The algorithm is parameter-free and guided by the optimization of the VAC. First, our algorithm automatically performs a coarse approximation to the correct number of clusters such that the robust fitting technique together with the filterOpt algorithm succeeds in determining the correct model of the clusters and purifying them from noise. We apply a simple top-down splitting technique similar to Pelleg and Moore [2000], but we additionally apply the filterOpt algorithm and keep the result if purifying reduces the VAC.

We start with the whole dataset as one cluster. The filterOpt algorithm is applied to the whole dataset and the VAC of the result is saved. Then the first split is performed. A split of a cluster C comprises the following steps: We run ordinary K-Means with $k = 2$ to get an approximate partitioning of C into two intermediate clusters C_i and C_j. On each of these two clusters, the filterOpt algorithm is applied. For example, if we consider cluster C_i and let C_{iC} be the cluster containing the core points of C_i, and C_{iN} be the cluster containing the noise points, then we only split up C_i into C_{iC} and C_{iN} if $VAC(C_{iC}) + VAC(C_{iN}) < VAC(C_i)$, otherwise C_i remains unchanged. The same is done for C_j. Thus, the result of a split of a cluster C consists of two clusters, each of which may be further divided into core and noise clusters. The sum of the VAC of these children is now compared to the VAC of C, and if an improvement has been achieved, the split is accepted.

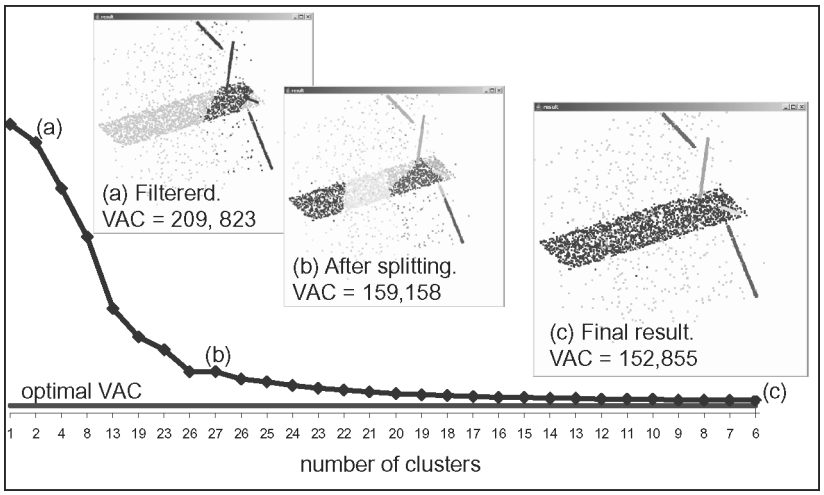


Fig. 15. Top-down algorithm on 3D dataset.

```
algorithm TopDown (data set DS, int t)
//Splitting.
  while savedCost > 0 and counter < t
  for each C ∈ 𝒞
    split(C);
//Merging.
ClusterMerging(clusters 𝒞, int t);
return Clustering with minimum overall VAC;

algorithm split(cluster C)
  set of clusters result;
  clusters C_i and C_j = 2-Means(C);
  run FilterOpt on C_i and C_j with available PDF, select result with lowest VAC;
  if VAC(C_*C ∪ C_*N) < VAC(C_*)
    result.add(C_*C, C_*N), else result.add(C_*);
  return result;
```

Fig. 16. Top-down algorithm.

If at least for one cluster an improvement can be achieved, we continue splitting these clusters. For the next split, we consider each cluster C_i including its core points and noise points as a single cluster. To avoid getting stuck in a local cost minimum we continue splitting all clusters for another t iterations, even if the VAC increases. Whenever a new cost minimum is reached, the counter is reset to 0. Finally, we try to split all clusters and noise clusters of the last cost minimum separately.

For cluster merging, the algorithm described in Section 3.3 can be used. As a more efficient alternative, in each step an arbitrary pair of clusters C_i, C_j can be considered. If $VAC(C_i \cup C_j) < VAC(C_i) + VAC(C_j)$, then C_i and C_j are merged. On our synthetic examples, the greedy merging algorithm described in Section 3.3 and this procedure achieve similar results. Figure 16 provides pseudocode for the top-down algorithm.

Experiments. Figure 15 illustrates the top-down algorithm on the 3D synthetic dataset first presented in Figure 7. Applying the filterOpt algorithm to the whole dataset results in two clusters (one with the core points, one noise cluster) and a VAC of 209,823 item (a) in the figure). The dataset is then split as long as splitting leads to an improvement of the VAC. At point (b) in the figure, we obetain the the maximum number of 27 clusters with a VAC of 159,158. Merging these clusters finally results in a clustering with 6 clusters and a VAC of 152,855, which is very close to the VAC of the ideal clustering of 151,637. Thus, the top-down algorithm performs better than both RIC after K-Means (153,393) and RIC after DBSCAN (155,412).

Also on the 2D synthetic dataset we obtain a better result using the top-down algorithm. The result of RIC after K-Means with a VAC of 76,940 is depicted in Figure 6(c). Mainly due to improved filtering of the Laplacian cluster, we achieve a VAC of 76,309 with the top-down algorithm.

6. CONCLUSIONS

The contributions of this work are the answers to the two questions we posed in the introduction, organized in our RIC framework.

—*(Q1) Goodness Measure.* We propose the VAC criterion using information-theory concepts, and specifically the volume after compression.

—*(Q2) Efficiency.* We introduce two novel algorithms, which together can help us find good groupings in a fast, "greedy" fashion:
  —the robust fitting (RF) algorithm, which carefully avoids outliers. Outliers plague all methods that use the Euclidean distance (or, equivalently, try to maximize the likelihood for Gaussian clusters).
  —the cluster merging (CM) algorithm, which stitches clusters together if the stitching gives a better VAC score.

We show that our RIC framework is very flexible, with several desirable properties that previous clustering algorithms don't have, summarized next.

—It can handle any of the known distributions (Gaussian, uniform, Laplacian). The vast majority of clustering algorithms focus on Gaussian distributions only.

—It can be extended to any other distribution we want.

—It is orthogonal to the searching algorithm that will look for clusters.

—It naturally gives outliers (single-member clusters).

—It gives more information; not only does it give the clusters, but also the cluster shapes (uniform, Gaussian, Laplacian).

—It is fully automatic (no complex parameter setting) and time and space efficient.

More importantly, the RIC framework does *not* compete with existing (or future) clustering methods: In fact, it can benefit from them! If a clustering algorithm is good, our RIC framework will use its grouping as a starting point, then try to improve on it (through the robust fitting and cluster merging algorithms),

either doing so, or declaring it as the winner. In short, the RIC framework *cannot lose*—at worst, it will tie!

We also presented experiments on real and synthetic data, where we showed that our RIC framework and algorithms give intuitive results while typical clustering algorithms fail.

REFERENCES

AGGARWAL, C. C. AND YU, P. S. 2000. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 70–81.

AGRAWAL, R., GEHRKE, J., GUNOPULOS, D., AND RAGHAVAN, P. 1998. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 94–105.

ANKERST, M., BREUNIG, M. M., KRIEGEL, H.-P., AND SANDER, J. 1999. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the ACM International SIGMOD Conference on Management of Data* .

BANFIELD, J. D. AND RAFTERY, A. E. 1993. Model-based Gaussian and non-Gaussian clustering. *Biometrics 49*, 3, 803–821.

BHATTACHARYA, A., LJOSA, V., PAN, J.-Y., VERARDO, M. R., YANG, H., FALOUTSOS, C., AND SINGH, A. K. 2005. ViVo: Visual vocabulary construction for mining biomedical images. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*.

BÖHM, C., KAILING, K., KRÖGER, P., AND ZIMEK, A. 2004. Computing clusters of correlation connected objects. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 455–466.

CHAKRABARTI, D., PAPADIMITRIOU, S., MODHA, D. S., AND FALOUTSOS, C. 2004. Fully automatic cross-associations. In *Proceedings of the ACM SIGKDD Conference on International Knowledge Discovery and Data Mining*. 79–88.

ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the ACM SIGKDD Conference on International Knowledge Discovery and Data Mining* .

GRÜNWALD, P. 2005. A tutorial introduction to the minimum description length principle. *Advances in Minimum Description Length: Theory and Applications*.

GUHA, S., RASTOGI, R., AND SHIM, K. 1998. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 73–84.

HAMERLY, G. AND ELKAN, C. 2003. Learning the k in k-means. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*.

HARTIGAN, J. A. 1975. *Clustering Algorithms*. John Wiley.

JOLLIFFE, I. 1986. *Principal Component Analysis*. Springer.

MURTAGH, F. 1983. A survey of recent advances in hierarchical clustering algorithms. *Comput. J. 26*, 4, 354–359.

NG, A., JORDAN, M., AND WEISS, Y. 2001. On spectral clustering: Analysis and an algorithm. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*.

NG, R. T. AND HAN, J. 1994. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the Conference on Very Large Databases (VLDB)*, 144–155.

PELLEG, D. AND MOORE, A. 2000. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, 727–734.

SLONIM, N. AND TISHBY, N. 2000. Document clustering using word clusters via the information bottleneck method. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 208–215.

STILL, S. AND BIALEK, W. 2004. How many clusters? An information theoretic perspective. *Neural Comput. 16*, 2483–2506.

TIBSHIRANI, R., WALTHER, G., AND HASTIE, T. 2000. Estimating the number of clusters in a dataset via the gap statistic. Tech. Rep., Stanford University.

TISHBY, N., PEREIRA, F. C., AND BIALEK, W. 2000. The information bottleneck method. In *Proceedings of the 37th Allerton Conference on Communication, Control and Computing*.

TUNG, A. K., XU, X., AND OOI, B. C. 2005. CURLER: Finding and visualizing nonlinear correlation clusters. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 467–478.

VAN-RIJSBERGEN, C. 1979. *Information Retrieval*, 2nd ed. Butterworths, London.

ZHANG, B., HSU, M., AND DAYAL, U. 2000. K-harmonic means—A spatial clustering algorithm with boosting. In *Proceedings of the 1st International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers (TSDM)*, 31–45.

ZHANG, T., RAMAKRISHNAN, R., AND LIVNY, M. 1996. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM International SIGMOD Conference on Management of Data*, 103–114.

Received December 2006; accepted July 2007