Robust Information-theoretic Clustering

Christian Böhm [1], Christos Faloutsos [2], Jia-Yu Pan [2], Claudia Plant [3]
[1]: University of Munich, Germany, [2]: CMU, Pittsburgh, PA, USA, [3]: UMIT, Hall, Austria
[1]: boehm@ifi.lmu.de, [2] : {christos, jypan}@cs.cmu.edu, [3]: claudia.plant@umit.at

ABSTRACT

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? Most clustering algorithms were designed with certain assumptions (Gaussianity), they often require the user to give input parameters, and they are sensitive to noise. In this paper, we propose a robust framework for determining a natural clustering of a given data set, 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. In fact, RIC is even able to purify and improve an initial coarse clustering, even if we start with very simple methods such as grid-based space partitioning. Moreover, RIC scales well with the data set size. Extensive experiments on synthetic and real world data sets validate the proposed RIC framework.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining

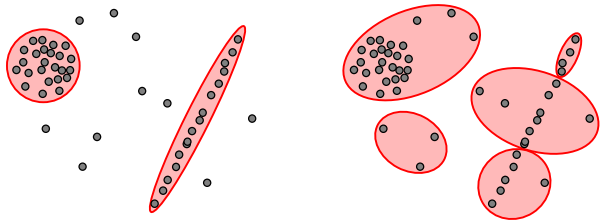
General Terms

Algorithms, Performance

Keywords

Clustering, Noise-robustness, Parameter-free Data Mining, Data Summarization

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.



(a) 'good' (b) 'bad'

Figure 1: A fictitious dataset, (a) with a good clustering of one Gaussian cluster, one sub-space cluster, and noise; and (b) a bad clustering.

1. INTRODUCTION

The problem of clustering has attracted a huge volume of attention for several decades, with multiple books [11, 20], surveys [13] and papers (X-means [16], G-means [10], CLARANS [15], CURE [9], CLIQUE [2], BIRCH [22], DBSCAN [7], to name a few). Recent interest in clustering has been on finding clusters that have non-Gaussian correlations in subspaces of the attributes, e. g. [5, 19, 1]. Finding correlation clusters has diverse applications ranging from spatial databases to bio-informatics. 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 from clustering.

For example, in Figure 1, we show a fictitious set of points in 2-d. 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 we try to answer in this work:

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 score to the one 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 overview and contributions, of this paper, are exactly the answers to the above two questions: For the first, we propose to envision the problem of clustering as a compression problem and use information-theoretic arguments. The grouping of Figure 1(a) is 'good', because it can succinctly describe the given dataset, with few exceptions: 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; the bytes to record their type (Gaussian, line-like, or something else, from a fixed vocabulary of distributions); the bytes 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.

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 paper: We propose to start from a sub-optimal grouping (e.g., using K-means, with some arbitrary k). Then, we propose to use two novel algorithms:

- *Robust fitting (RF)*, instead of the fragile PCA, to find low-dimensionality sub-space clusters and
- *Cluster merging (CM)*, to stitch promising clusters together.

We continue fitting and merging, until our VAC criterion reaches a plateau. The sketch of our algorithm above 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 Fit, and Cluster Merge). The resulting method has the following advantages:

1. It is fully automatic, i.e. no difficult or sensitive parameters must be selected by the user.
2. It returns a natural partitioning of the data set, thanks to the intuitive information theoretic principle of maximizing the data compression.
3. 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), and combinations and mixtures of clusters of all different types during one single run of the algorithm.
4. It can assign model distribution functions such as uniform, Gaussian, Laplacian (etc.) distribution to the different subspace coordinates and gives thus a detailed description of the cluster content.
5. It is robust against noise. Our Robust Fitting (RF) method is specifically designed to spot and ignore noise points.
6. It is space and time efficient, and thus scalable to large data sets.

To the best of our knowledge, no other clustering method meets all of the above properties. The rest of the paper 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 and Section 5 concludes our paper.

2. SURVEY

As mentioned, clustering has attracted a huge volume of interest over the past several decades. Recently, there are several papers focusing on scalable clustering algorithms, e. g. CLARANS [15], CURE [9], CLIQUE [2], BIRCH [22], DBSCAN [7] and OPTICS [3]. There are also algorithms that try to use no user-defined parameters, like X-means [16] and G-means [10]. However, they all suffer from one or more of the following drawbacks: they focus on spherical or Gaussian clusters, and/or they are sensitive to outliers, and/or they 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 [22] (which is suitable for spherical clusters), X-means [16] and G-means [10]. 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 also for OPTICS to derive clusters from the reachability-plot. K-harmonic means [21] avoids the problem of outliers, but still needs k. Spectral clustering algorithms [14] 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 [5, 19, 1]. Finding correlation clusters has diverse applications ranging from spatial databases to bio-informatics.

Parameter-free methods: A disproportionately small number of papers has focused on the subtle, but important problem of choosing k, the number clusters to shoot for. Such methods include the above mentioned X-means [16] and G-means [10], 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 un-natural, 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 the Information Bottleneck Method [18], which is used by Slonim and Tishby for clustering terms and documents [17]. Based on information theory they derive a suitable distance function for co-clustering, 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) [8]. 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 it 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 noise-free mixture of Gaussians. The difference is that our RIC can also handle noise, as well as additional data distributions (uniform, etc.).

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 hyper-plane to project the data onto, so that the Euclidean projection error is minimized. That is, PCA finds global correlation structures of a data set [12]. Recent work have extended PCA to identify local correlation structures that are linear [5] or nonlinear [19], however, some method-specific parameters such as neighborhood size or the dimensionality of microclusters, are still required. It is desirable to have a method that is efficient and robust to outliers, minimizing the need of pre-specified parameters.

3. PROPOSED METHOD

The quality of a clustering usually depends on noise in the data set, wrong algorithm parameters (e.g., number of clusters), or limitations on the method used (e.g., unable to detect correlation clusters), and resulting in a un-natural partition of the data set. Given an intial clustering of a data set, how do we systematically adjust the clustering, overcome the influence of noise, recognize correlation patterns for cluster formation, and to 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 data set. 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 presense 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 the criterion VAC are described in details in the following subsections. Table 1 gives a list of symbols used in this paper.

Symbol	Definition
VAC	Volume After Compression
RF	Robust Fit
CM	Cluster Merge
RIC	Robust Information-based Clustering
n	The number of points in the data set.
d	The dimensionality of the data set.
$\mathcal{C}$	The clusters of a data set, $\mathcal{C}=\{C_i \mid i = 1,\ldots,k\}$.
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 i-th 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.

Table 1: Table of Symbols and Acronyms

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 the number of clusters k, the encoding for the shape of each cluster (e.g., mean and covariance, if it is a Gaussian cluster), the encodings for the cluster-id and the (relative) coordinates of the data points.

We assume that all coordinates are integers, since we have finite precision, anyway. That is, we assume that our data points are on a d-dimensional grid. The resolution of the grid can be chosen arbitrarily.

The description of the method consists of the following parts: (a) how to encode integers (b) how to encode the points, once we determine that they belong in a given cluster.

The idea is best illustrated with an example. Suppose we have the dataset of Figure 1. Suppose that the available distributions in our RIC framework are two: Gaussian, and uniform (within a Minimum Bounding Rectangle). 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 [6], where integer i is represented using $O(\log i)$ bits. As Table 2 shows, we can encode the length of the integer in unary (using $\log i$ zeros), and then the actual value, 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.

number	coding	
	length	value
1	0	1
2	00	10
3	00	11
8	0000	1000

Table 2: Self-delimiting integer coding

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 if 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 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 if the points belong to the cluster subspace and to the characteristic distribution functions of the cluster. Later, we will inversely define that probability density function which gives the highest compression rate to be the right choice. 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. 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 code patterns to the coordinate values such that coordinate values with a high probability (such as $3 < x < 4$) are assigned short patterns, and coordinate values with a low probability (such as $y = 12$ to give a more extreme example) are assigned longer patterns. Provided that a coordinate is really

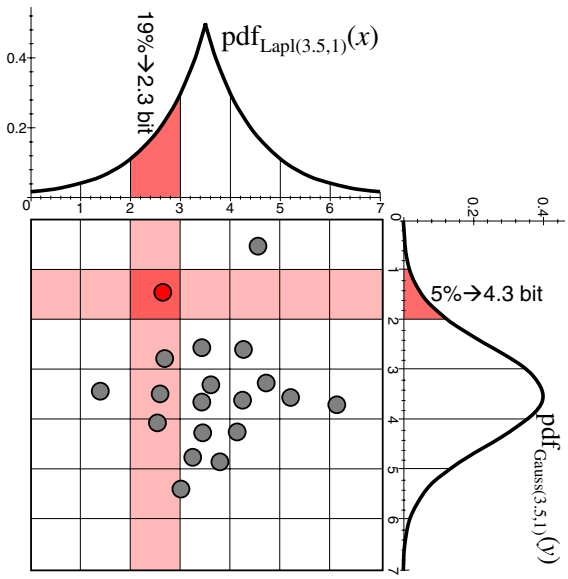


Figure 2: Example of VAC.

distributed according to the assumed distribution function, Huffman codes optimize the overall compression of the data set. 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 (discretized) 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, i.e. $PDF = \{pdf_{Gauss(\mu_i,\sigma_i)}, pdf_{uniform(lb_i,ub_i)}, pdf_{Lapl(a_i,b_i)}, ...\}$, $\mu_i, lb_i, ub_i, a_i \in \mathbb{R}, \sigma_i, b_i \in \mathbb{R}^+$. 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*

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

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

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

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.

Naturally, the question arises to what extent this coding depends on the choice of the grid resolution. The absolute value of the code length clearly depends on the grid resolution. 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. Since we only compare the VAC of different cluster structures, distribution functions, subspaces, etc, and leave the grid resolution at a constant level, high enough to distinguish the different points from each other, the overall result of our algorithm is not sensitive to the grid resolution.

Next, we address the question, 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 that *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 data set. Therefore, if the Gaussian *pdf* is selected for an attribute i, we use the mean and variance of the i-th 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, i.e.:

DEFINITION 2 (CHARACTERISTIC $\overrightarrow{pdf}(\vec{x})$ OF CLUSTER C). *Let C be a cluster with points $\vec{x} \in C$.*
Let $stat = (\mu_i, \sigma_i, lb_i, ub_i, ...)$ 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

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

For the x-coordinate of the example in Figure 2 that means the following: First required statistics, i.e. mean (3.5), variance (1.0), and lower and upper limit (1.4, 6.2) of the data set is determined. Then, VAC_x is determined for all allowed $pdf \in PDF$, i.e. 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 paper 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, i.e. if one (or more) attribute value of the points of the cluster depends on the value of one (or more) other attribute. 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^{T} gained from PCA diagonalization $V \Lambda V^{\mathrm{T}} = \Sigma$. We give more details on estimating the covariance matrix in a noise robust way 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 bit). 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, i.e. for each $\vec{y} \in Y : \vec{y} = V^{\mathrm{T}} \cdot \vec{x}$. 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 of C 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 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 the 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 reason 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 the 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 lead to a biased estimation of the cluster model.

We propose an algorithm for purifying a cluster that, after the processing, noise points are separated from their original cluster and form a cluster of their own. We start with

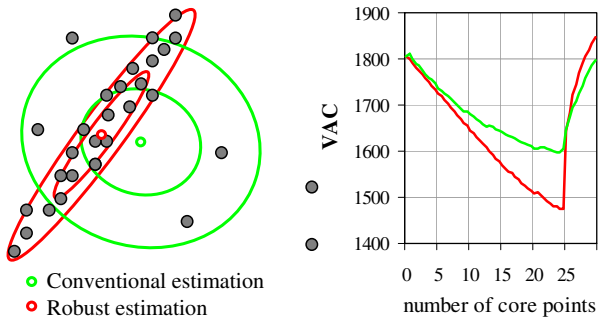


Figure 3: Conventional and robust estimation.

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 *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 the 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 the cluster and stored in a new cluster. Next, we elaborate on the steps for purifying a cluster of points.

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 ($\Sigma = V\Lambda V^{\mathrm{T}}$), 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:

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

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 following averaging:

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

where $(\vec{x}-\vec{\mu}) \cdot (\vec{x}-\vec{\mu})^{\mathrm{T}}$ 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 i-th and j-th attributes, which is the product of the attribute values $(x_i-\mu_i)\cdot(x_j-\mu_j)$, averaged over all data points $\vec{x} \in C$. 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, i.e. 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 the conventional estimation. In addition, the ellipsoid which shows the estimated "data spread" corresponding to the covariance matrix has a completely wrong direction which 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 attribute independently. The result is a data set where the origin is close to the center of the 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 the 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 the 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%):

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

It can easily be seen that adding the matrix ϕI does only affect the Eigenvalues and not the Eigenvectors: If $\Sigma = V\Lambda V^{\mathrm{T}}$ then $\Sigma + \phi I = V\Lambda V^{\mathrm{T}} + \phi I$. Since V is orthonormal, ϕI can also be written as $V\phi I V^{\mathrm{T}}$, and due to the distributive law we have $\Sigma + \phi I = V(\Lambda + \phi I)V^{\mathrm{T}}$, i.e. 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 the core points. The safe decorrelation matrix V (cf. 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, i.e. no information loss.

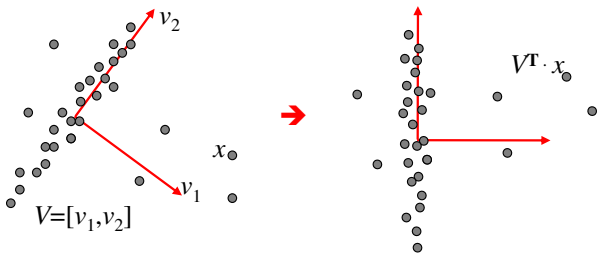


Figure 4: The decorrelation matrix.

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, matrix Σ_R from the robust method described above. 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 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 the 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 had the least overall cost in this algorithm is finally returned (cf. Figure 3 where at the minimum (1480) we have 24 objects in the core set and 6 objects in the noise set). The diagram in Figure 3 depicts the VAC value (Y-axis) of the different configuration (C_{core}, C_{out}) at each iteration (X-axis). Figure 3 shows

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;

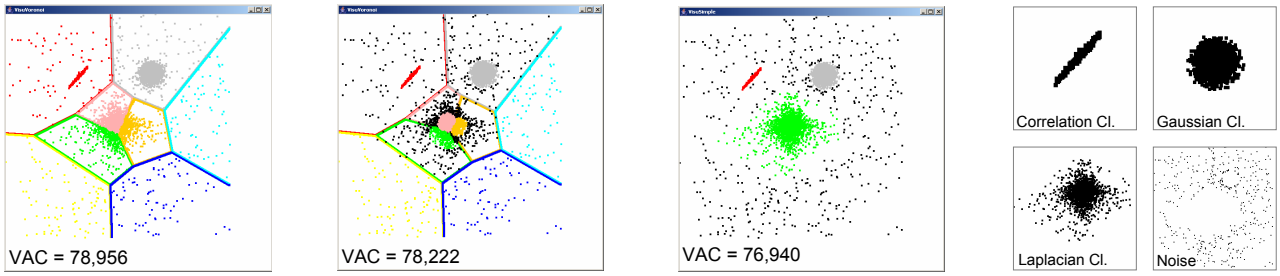
subroutine $[(C*_1, C*_2)$, mergedCost$(C*_1,C*_2)]$= findMergePair(clusters $\mathcal{C}$)
// Find cluster pair with the best (maximal savedCost).
for all cluster pairs $(C_i, C_j) \in \mathcal{C} \times \mathcal{C}$
mergedVAC(C_i,C_j) := VAC$(C_i \cup C_j)$;
savedCost(C_i,C_j):=(VAC(C_i)+VAC(C_j))-mergedVAC(C_i,C_j);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

Figure 5: RIC algorithm.

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.

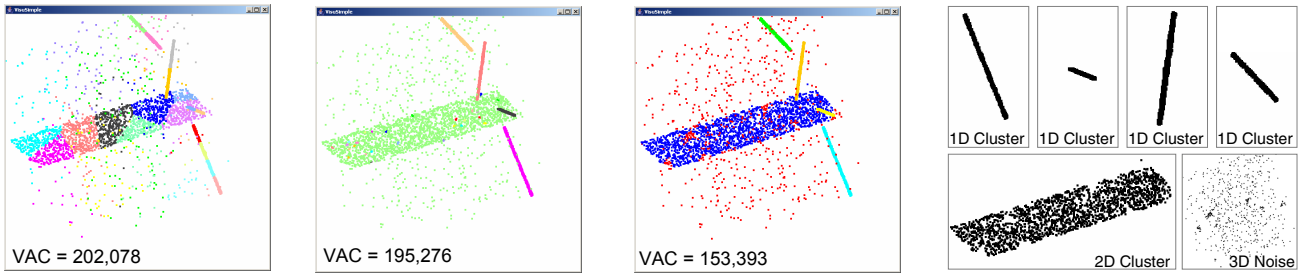
3.3 Cluster Merging (CM)

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 proce-



(a) K-means. (b) After purifying. (c) RIC result. (d) Detailed view.

Figure 6: 2-d synthetic data.



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

Figure 7: 3-d synthetic data.

dure 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 non-compact. 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, takes 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 the two clusters which have the maximum $savedCost(.,.)$ value, resulting in a greedy search toward a clustering that has the minimum overall cost. To avoid this greedy algorithm from getting stuck in a local minimum, we do not stop immediately, even when there is no saving of $savedCost(.,.)$ value can be achieved by merging pairs of clusters. That is, we do not stop when $savedCost(.,.) \leq 0$. Instead, the algorithm continues for another t iterations, continuous 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 data set. Whenever a new minimum is reached the counter is reset to zero. Pseudocode for the RIC algorithm is given in Figure 5.

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 2-d data set consisting of 4751 data objects. Two of the resulting clusters contain many noise objects, among them the one dimensional 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 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, supports subspace and correlation clusters and 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 clusters with common characteristics, such as common subspace orientation or 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 (cf. 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 value-added 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 coordinates of the bottom right cluster (representing the noise objects) are uniformly distributed.

We demonstrate the performance of the cluster filtering and merging algorithm on a 3-d synthetic data containing 7500 data objects (cf. Figure 7). This data set consists of one plane (2-d correlation cluster, 2000 objects) and 3 lines (1-d correlation clusters, two with 2000 objects each, one with 1000 objects) 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 are 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. 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.

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 (cf. 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 7(c). This demonstrates that the RIC framework can be applied with various partitioning clustering methods. Since the data set 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 gready fashion optimization process is efficient. We implemented the RIC algorithm in Java. Runtimes for the synthetic data sets are 147 s for the 2-d data set and 567 s for the 3-d data set on a PC with 3 GHz CPU and 1 GB RAM.

4.2 Performance on Real Data

4.2.1 Metabolome Data

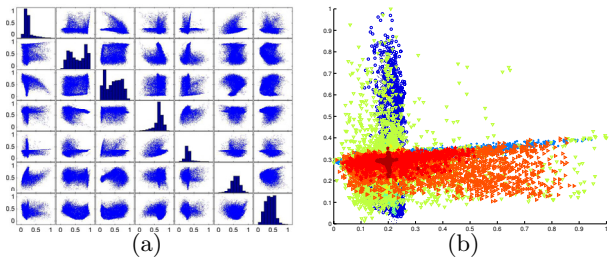
We evaluate the RIC framework using a high dimensional metabolic data set. This 14-dimensional data set (643 instances) was produced by modern screening methodologies and represents 306 cases of PKU, a metabolic disorder, and 337 objects from a healthy control group. As initial clusterings, we used spectral 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 3 shows the the same quantities and the clustering results after we apply RIC. Notice that in all cases, RIC achieved everything we wanted: (a) it found the correct number of clusters, (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 to them over RIC). Even so, notice that RIC achieves both lower VAC score, as well as better *impurity* count IMP. Using $k = 2$, both, K-means and spectral clustering assign many instances of class PKU to the cluster of the control group.

Table 3: 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

4.2.2 Cat Retina Images

The data we considered here are image blocks in retinal images from the UCSB BioImage (http://bioimage.ucsb.edu/) database. The blocks are taken from 219 images of retina under 9 different conditions (healthy, diseased, etc.). Each image is of size 512-by-768. We take non-overlapped pixel blocks (which are called *tiles*) of size 64-by-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,



(a) (b)

Figure 8: (a): Visualizing the distribution of the 7-dimensional retinal image tiles. Each subfigure shows the distribution of two dimensions. The data set contains non-Gaussian clusters. (b): The 13 clusters found by RIC. Figures look best in color.

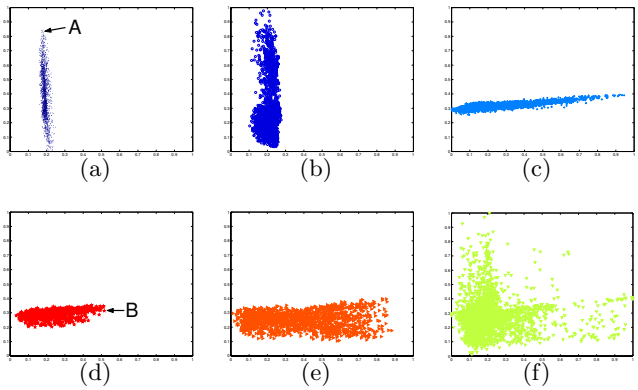


Figure 9: Example clusters on retinal image tiles found by RIC.

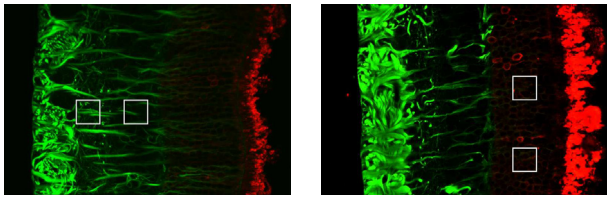


Figure 10: The white boxes in the two retinal images indicate example tiles in selected clusters. Left (a): Tiles at position A of cluster of Figure 9(a) Right (b): Tiles at position B of cluster of Figure 9(d). Best viewed in color.

exactly as suggested in [4]. Figure 8(a) visualizes the distribution of the image tiles. The distribution is viewed from all possible pairs of dimensions – the (i, j)-subfigure plots the i-th dimension versus the j-th dimension. 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 first and 5-th dimensions (the subfigure at the first row and the fifth column). In the following discussion, we will focus on the view of the first and 5-th dimensions, and show that our RIC framework is able to find the non-Gaussian, correlation clusters in this data set.

Moreover, most of the coordinates in the detected clusters clearly show a supergaussian 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 data set. We plot each cluster separately in different figures, for better visualization of the clustering result.

Some plots of individual clusters are shown in Figure 9(a)-(f). It can be easily seen that the proposed RIC method successfully finds the correlation clusters in this data set, and, unlike other methods like K-means, it will neither over-cluster nor under-cluster the data set.

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 red and green 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. The tiles highlighted 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 3 months. The tiles highlighted are the "rod photoreceptors", with the protein rod opsin redistributed into the cell bodies, which are 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. 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 the 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:

- it can handle any of the known distributions (Gaussian, uniform, Laplacian) The vast majority of clustering algorithms focus on the 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 it gives 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, it will try to improve on it (through the 'Robust Fit' and 'Cluster Merge' algorithms), and, it will either improve it, or declare it as the winner. In short, the RIC framework *can not 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.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. IIS-0209107, SENSOR-0329549, EF-0331657, IIS-0326322, 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 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.

7. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD Conference*, pages 70–81, 2000.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD Conference*, 1999.

[4] A. Bhattacharya, V. Ljosa, J.-Y. Pan, M. R. Verardo, H. Yang, C. Faloutsos, and A. K. Singh. ViVo: Visual vocabulary construction for mining biomedical images. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM)*, 2005.

[5] C. Böhm, K. Kailing, P. Kröger, and A. Zimek. Computing clusters of correlation connected objects. In *SIGMOD Conference*, pages 455–466, 2004.

[6] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD Conference*, pages 79–88, 2004.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD Conference*, 1996.

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

[9] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *SIGMOD Conference*, pages 73–84, 1998.

[10] G. Hamerly and C. Elkan. Learning the k in k-means. In *Proceedings of NIPS*, 2003.

[11] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, 1975.

[12] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[13] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[14] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001.

[15] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of VLDB Conference.*, pages 144–155, 1994.

[16] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 727–734, 2000.

[17] N. Slonim and N. Tishby. Document clustering using word clusters via the information bottleneck method. In *SIGIR*, pages 208–215, 2000.

[18] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *CoRR*, physics/0004057, 2000.

[19] A. K. Tung, X. Xu, and B. C. Ooi. CURLER: Finding and visualizing nonlinear correlation clusters. In *SIGMOD Conference*, pages 467–478, 2005.

[20] C. Van-Rijsbergen. *Information Retrieval*. Butterworths, London, England, 2nd edition, 1979.

[21] B. Zhang, M. Hsu, and U. Dayal. K-harmonic means - A spatial clustering algorithm with boosting. *TSDM*, pages 31–45, 2000.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD Conference*, pages 103–114, 1996.