Technische
Universität
Wien

Vienna
University of
Technology

Institut f. Statistik u. Wahrscheinlichkeitstheorie

A toolbox for k-centroids cluster analysis

F. Leisch

Forschungsbericht CS-2005-2

September 2005

1040 Wien, Wiedner Hauptstr. 8-10/107
AUSTRIA

Email: P.Filzmoser@tuwien.ac.at
http://www.statistik.tuwien.ac.at

A Toolbox for K-Centroids Cluster Analysis

Friedrich Leisch

Department of Statistics and Probability Theory
Vienna University of Technology, 1040 Vienna, Austria
http://www.ci.tuwien.ac.at/~leisch

This is a preprint of an article accepted for publication in:
Computational Statistics and Data Analysis, 2006.
http://www.elsevier.com/locate/csda

Abstract

A methodological and computational framework for centroid-based partitioning cluster analysis using arbitrary distance or similarity measures is presented. The power of high-level statistical computing environments like R enables data analysts to easily try out various distance measures with only minimal programming effort. A new variant of centroid neighborhood graphs is introduced which gives insight into the relationships between adjacent clusters. Artificial examples and a case study from marketing research are used to demonstrate the influence of distances measures on partitions and usage of neighborhood graphs.

1 Introduction

New developments for partitioning cluster analysis have been dominated by algorithmic innovations over the last decades, with the same small set of distance measures used in most of them. Especially the machine learning literature is full of "new" cluster algorithms for Euclidean or—to much lesser extent—L^1 distance (also called Manhattan distance). For hierarchical cluster analysis it has always been natural to treat distance (or similarity measures) and details of the cluster algorithm like the linkage method at par, and most software implementations reflect this fact, offering the user a wide range of different distance measures. In programmatic environments like S (Becker et al., 1988) hierarchical clustering often works off a distance matrix, allowing for arbitrary distance measures to be used.

Most monographs on cluster analysis do the same for partitioning cluster analysis and state the corresponding algorithms in terms of arbitrary distance measures, see e.g., Anderberg (1973) for an early overview. However, partitioning cluster analysis has often been the method of choice for segmenting "large" data sets, with the notion of what exactly a "large" data set is changing considerably over time. So computational efficiency always was an important consideration for partitioning cluster algorithms, narrowing the choice of distance methods to those where closed-form solutions for cluster centroids exist: Euclidean and Manhattan distance. Other distance measures have been used for the task, of course, but more often than not "new" cluster algorithms got invented to deal with them.

The goal of this paper is to shift some of the focus in partitioning cluster analysis from algorithms to the distance measure used. Algorithms have an important influence on clustering (convergence in local optima, dependency on starting values, etc.), but their efficiency should no longer be the main consideration given modern computing power. As an example for an "unusual" distance measure, Heyer et al. (1999) use the so-called jacknife correlation for clustering gene expression profiles, which is the average correlation after removing each time point once.

Many other distance measures could be useful in applications if their usage does not put too much computational burden like programming effort onto the practitioner.

This paper is organized as follows: In Section 2 we define the notation used throughout the paper, give a general formulation of the K-centroids cluster analysis (KCCA) problem and write several popular cluster algorithms for metric spaces as special cases within this unifying framework. Section 3 shows how one can adapt the framework to arbitrary distance measures using examples for clustering nominal spaces, and especially binary data. Section 4 describes a flexible implementation of the framework that can be extended by users with only minimal programming effort to try out new distance measures. Section 5 introduces a new version of cluster neighborhood graphs which allows for visual assessment of the cluster structure. Finally, Section 6 demonstrates the methods on a real data set from tourism marketing.

2 The K-Centroids Cluster Problem

2.1 Distances, Similarities and Partitions

Let $\mathcal{X}$ denote a space of deterministic or random variables, μ a (probability) measure on the Borel sets of $\mathcal{X}$, $\mathcal{C}$ a set or space of admissible centroids, and let

$$d(\mathbf{x}, \mathbf{c}): \quad \mathcal{X} \times \mathcal{C} \to \mathbb{R}^+$$

denote a distance measure on $\mathcal{X} \times \mathcal{C}$. A set of K centroids

$$C_K = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}, \qquad \mathbf{c}_k \in \mathcal{C}$$

induces a *partition* $\mathcal{P} = \{\mathcal{X}_1, \ldots, \mathcal{X}_K\}$ of $\mathcal{X}$ into K disjoint clusters by assigning each point to the segment of the closest centroid

$$\begin{aligned} \mathcal{X}_k &= \{\mathbf{x} \in \mathcal{X} | c(\mathbf{x}) = \mathbf{c}_k\}, \\ c(\mathbf{x}) &= \operatorname*{argmin}_{\mathbf{c} \in C_K} d(\mathbf{x}, \mathbf{c}). \end{aligned}$$

If we use a similarity measure $s(\mathbf{x}, \mathbf{c}) \in [0, 1]$ instead of a distance measure we get a partition by defining

$$c(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{c} \in C_K} s(\mathbf{x}, \mathbf{c}).$$

Note that a similarity measure can easily be turned into a distance measure (and vice versa), e.g., the transformation

$$d(\mathbf{x}, \mathbf{c}) = \sqrt{1 - s(\mathbf{x}, \mathbf{c})}$$

turns any non-negative definite matrix of similarities into a matrix of so-called Euclidean distances (Everitt et al., 2001). A matrix of pairwise (arbitrary) distances of objects is called Euclidean if the objects can be represented by points in a Euclidean space which have the same pairwise distances as the original objects, which is an appealing property, especially for visualization. For notational simplicity we will use only distances in the following, all algorithms can also be written for similarities (basically all minimization problems are replaced by maximizations).

Definition: The set of centroids C_K is called *canonical* for $(\mathcal{X}, \mathcal{C}, d)$, if

$$\int_{\mathcal{X}_k} d(\mathbf{x}, \mathbf{c}_k)\, d\mu(\mathbf{x}) \leq \int_{\mathcal{X}_k} d(\mathbf{x}, \mathbf{c})\, d\mu(\mathbf{x}) \qquad \forall \mathbf{c} \in \mathcal{C} \qquad \forall \mathcal{X}_k \in \mathcal{P}$$

where $\mathcal{P}$ is the partition induced by C_K.

2.2 The Cluster Problems

Usually we want to estimate a "good" set C_K of centroids from a data set $X_N = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_n \in \mathcal{X}$. The **$K$-centroids cluster problem** is to find a set of centroids C_K for fixed K such that the average distance of each point to the closest centroid is minimal

$$D(X_N, C_K) = \frac{1}{N} \sum_{n=1}^{N} d(\mathbf{x}_n, c(\mathbf{x}_n)) \to \min_{C_K},$$

i.e., to find the K canonical centroids for $(X_N, \mathcal{C}, d)$.

Recently a related problem has emerged in the bioinformatics literature (Heyer et al., 1999; Smet et al., 2002) where the number of clusters K is not fixed, because the maximum radius of the clusters is the principal quantity of interest. The **maximum radius cluster problem** is to find the minimal K such that a set of centroids C_K exists where

$$\max_{n=1,\ldots,N} d(\mathbf{x}_n, c(\mathbf{x}_n)) \leq r$$

for a given radius r. Another formulation of the problem is in terms of the maximum diameter of the clusters (maximum distance between two points), however by using the triangle inequality it can easily be shown that the two formulations are approximately the same:

$$\max_{n,m} d(\mathbf{x}_m, \mathbf{x}_n) \leq \max_{n,m} \left(d(\mathbf{x}_m, \mathbf{c}) + d(\mathbf{x}_n, \mathbf{c})\right) \leq 2 \max_{n} d(\mathbf{x}_n, \mathbf{c}), \quad \forall \mathbf{c}.$$

The two points with maximum distance in a cluster define its diameter, however the centroid of the cluster will usually not be equidistant to those two points. The radius gives an upper bound for the diameter, so minimizing the radius also decreases the diameter, but the global minima for the two problems will usually not be exactly the same.

Note that not every distance measure necessarily fulfills the triangle inequality. In this case the maximum radius problem and maximum diameter problem my have arbitrarily different solutions, depending on how strongly the triangle inequality is violated. For all other results throughout this paper the triangle inequality is not needed, i.e., d need not be a metric.

Representing clusters by centroids has computational advantages when predicting cluster membership for new data. For radius calculation one needs only comparison with the K centroids, whereas for diameter calculation one needs pairwise comparison with all N data points. We will not explicitly consider the maximum radius cluster problem for the rest of the paper, although most results (and our software implementation) can be used for it as well. A common problem for all K-centroids methods is that they assume a more or less equal within-cluster variability, which limits their use.

2.3 Parameter Estimation

2.3.1 Generalized K-Means

Even for very simple distance measures no closed form solution for the K-centroids cluster problem exists and iterative estimation procedures have to be used. A popular choice is to use the well-known K-means algorithm in its general form:

1. Start with a random set of initial centroids C_K (e.g., by drawing K points from X_N).
2. Assign each point $\mathbf{x}_n \in X_N$ to the cluster of the closest centroid.
3. Update the set of centroids holding the clusters $c(\mathbf{x}_n)$ fixed:

$$\mathbf{c}_k := \operatorname*{argmin}_{\mathbf{c} \in \mathcal{C}} \sum_{n: c(\mathbf{x}_n) = \mathbf{c}_k} d(\mathbf{x}_n, \mathbf{c}), \qquad k = 1, \ldots, K.$$

4. Repeat steps 2 and 3 until convergence.

Convergence: Both steps 2 and 3 of the algorithm do not increase the objective function $D(X_N, C_K)$, i.e., $D()$ is monotonely non-increasing during the iterations. Because only a finite number of possible partitions of the set X_N exists, the algorithm is guaranteed to converge in a finite number of iterations to a *local* optimum of the objective function. ◇

Although the algorithm was formulated using arbitrary distance measures from the beginning (e.g., MacQueen, 1967; Anderberg, 1973), it is somewhat synonym for partitioning clustering with respect to Euclidean distance. Many authors also use the term *K-means* for the general problem of finding canonical centroids with respect to Euclidean distance (e.g., Hartigan and Wong, 1979).

The algorithm converges only to the next *local* minimum of the objective function, and the usual recommendation is to try several starts with different random initializations and use the best solution. Numerous heuristics for "good starting values" have been published in the literature and are omitted here for simplicity. Hand and Krzanowski (2005) have recently proposed a variation of the basic scheme shown above which is inspired by simulated annealing: Instead of assigning each point to the cluster of the closest centroid some points are (with decreasing probability over the iterations) assigned to random other clusters. Experiments show that this helps in avoiding local minima.

2.3.2 Competitive Learning and Exchange Algorithms

Especially in the machine learning literature *online algorithms* are popular for clustering metric spaces (e.g., Ripley, 1996), this is also called **competitive learning**. The term "online" refers to iterative algorithms where data points are used one at a time as opposed to "offline" (or "batch") algorithms where each iteration uses the complete data set as a whole. Most algorithms of this type are a variation of the following basic principle: draw a random point from the data set and move the closest centroid(s) in the direction of this data point.

Another family of online algorithms are **exchange methods** like the one proposed by Hartigan and Wong (1979): Start with a random partition, consider a random data point $\mathbf{x}_n$ and re-assign it to a different cluster if that change improves the overall performance measure. Exchange methods have received a lot of attention in the statistical clustering literature, see e.g. Kaufman and Rousseeuw (1990) and references therein.

Both competitive learning and exchange algorithms have a serious limitation when used in highly vectorized interpreted languages like S, R or MATLAB: matrix calculations are typically efficiently implemented in compiled code, such that standard computations on blocks of data are fast (e.g., Section 2.7 in Venables and Ripley, 2000). K-means type algorithms can use all observations in one cluster as such a block of data and centroid computation can be vectorized. Online algorithms are genuinely iterative calculations and cannot be vectorized in most cases. They require the equivalent of several loops with one iteration each for every single observation. For this reason we consider only offline algorithms in the following which can be more easily extended using interpreted code only (no C, C++, or Fortran programming required).

2.4 Example: Clustering Metric Spaces

In m-dimensional metric spaces ($\mathbb{R}^m$ or subsets of it) the centroids are usually also vectors in $\mathbb{R}^m$, hence $\mathcal{X} = \mathcal{C} = \mathbb{R}^m$. The framework presented above contains the two most popular partitioning cluster algorithms as special cases:

K-means uses Euclidean distance $d(\mathbf{x}, \mathbf{c}) = ||\mathbf{x} - \mathbf{c}||$, the canonical centroids are the means of each cluster.

K-medians uses Manhattan distance $d(\mathbf{x}, \mathbf{c}) = |\mathbf{x} - \mathbf{c}|$, the canonical centroids are the medians of each cluster.

The existence of a closed form solution to find the canonical centroids makes these algorithms very fast. However, modern computing power enables us to approximate the centroids of each

cluster by using iterative optimization algorithms in reasonable amounts of time. The choice of a metric should be based on the data and application, not only on computational efficiency. Other possible choices for $d()$ include p-norms for arbitrary values of p

$$d(\mathbf{x}, \mathbf{c}) = \left(\sum_{i=1}^{m} |x_i - c_i|^p \right)^{1/p}$$

or the maximum norm $d(\mathbf{x}, \mathbf{c}) = \max_i |x_i - c_i|$. Kohonen (1984) proposes a vector quantization technique using the inner product between two vectors as similarity measure

$$s(\mathbf{x}, \mathbf{c}) = \frac{< \mathbf{x}, \mathbf{c} >}{||\mathbf{x}||||\mathbf{c}||}$$

which is equivalent to using the angle as distance. As the length of $\mathbf{c}$ has no influence on s, a common condition is to use only vectors of length 1 as centroids.

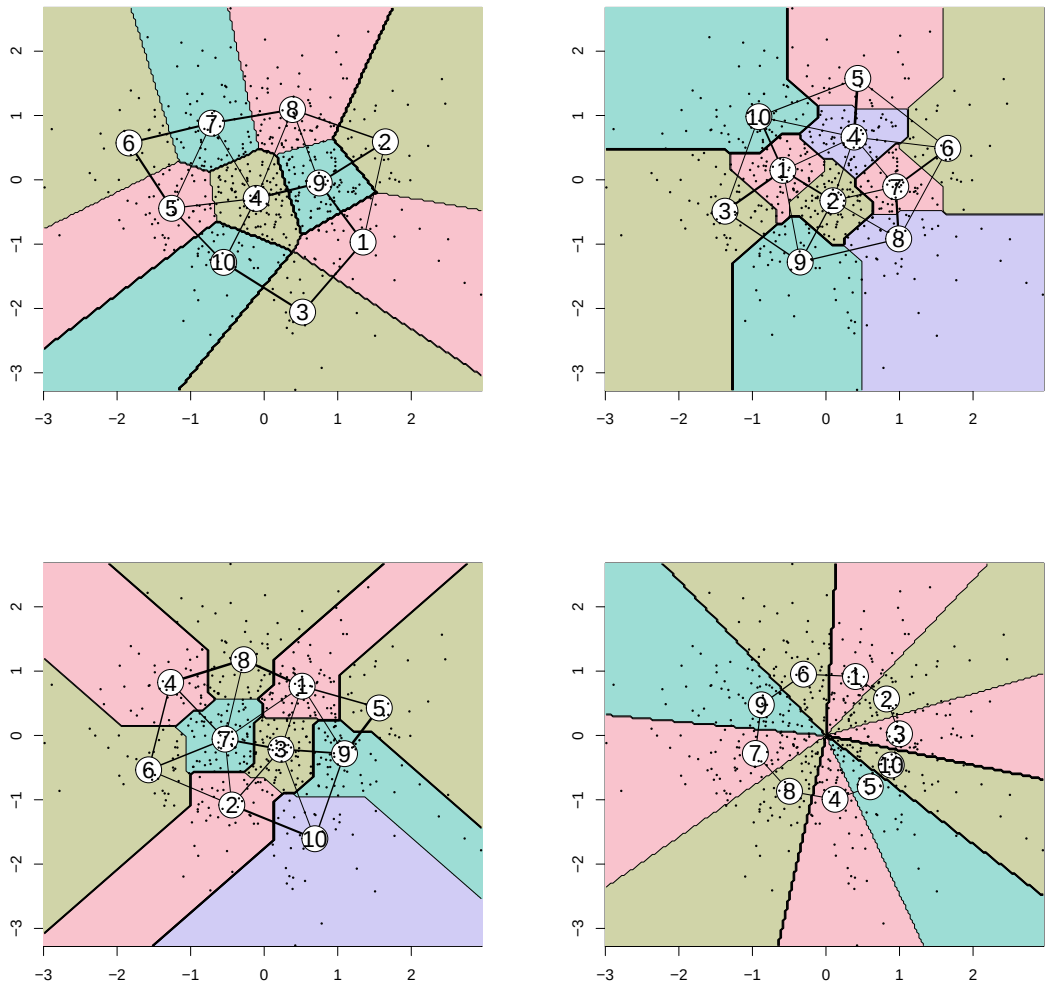


Figure 1: 500 2-dimensional standard normal data points clustered using Euclidean distance (top left), absolute distance (top right), maximum norm (bottom left) and angle (bottom right).

Figure 1 shows the different topologies that are induced by different distance measures on data without a cluster structure (the examples use 10 clusters on 500 points from a bivariate standard normal distribution). Partitioning data without any "natural clusters" into "arbitrary clusters" (Kruskal, 1977) may seem useless at first sight, see the marketing example in Section 6 or Everitt et al. (2001, p. 6ff) for applications.

All partitions have been estimated using the generalized K-means algorithm from Section 2.3.1. The locations of the centroids for 1-, 2- and maximum-norm are structurally not that different on this data set, however the partitions are. Euclidean distance gives approximately radial borders between the outer segments, absolute distance gives borders that are parallel to the axes, and maximum norm gives borders that are parallel to the main diagonals. Using angles results in centroids on the unit circle (for $||\mathbf{c}_k|| = 1$) and exactly radial segment borders.

Which type of partition is preferable depends on the application, but note that the by far most popular distance (Euclidean) results in the most irregular-shaped partitions, especially for "outer segments". In higher-dimensional spaces almost all clusters will be "outer segments" due to the curse of dimensionality, such that absolute distance might be preferable because the resulting segments are close to axis-parallel hypercubes and probably easier to interpret for practitioners.

3 Clustering Nominal Variables

3.1 The K-modes Algorithm

Chaturvedi et al. (2001) introduce a clustering algorithm for nominal data they call K-modes, which is a generalized K-means algorithm for a special distance measure: Suppose $\mathcal{X} = \mathcal{C}$ are m-dimensional spaces of nominal variables and we use the distance measure

$$d(\mathbf{x}, \mathbf{c}) = \frac{\#\{i|\, x_i \neq c_i\}}{m}$$

which counts in how many dimensions an observation $\mathbf{x}$ and centroid $\mathbf{c}$ do not have the same value. The canonical centroids are the dimension-wise modes of the points in each cluster, i.e., c_{k1} is set to the most frequent value of x_{n1} for all $\mathbf{x}_n$ where $c(\mathbf{x}_n) = k$, c_{k2} is set to the most frequent value of x_{n2}, etc., and ties are broken at random.

An obvious generalization of K-modes would be to specify m loss matrices $L^h = [l^h_{ij}]$, $h = 1, \ldots, m$, where l^h_{ij} is the loss occurring when $x_{nh} = i$ and $c_{kh} = j$. The canonical centroids then are (element-wise) set to the minimum-cost category instead of simply the most frequent category. For ordinal data one could also use the K-medians algorithm.

Listing all possible distance measures for nominal or mixed scale data is beyond the scope of this paper. E.g., the chapter on distance measures in Ramon (2002), which provides an excellent survey from a machine learning point of view, is about 80 pages, followed by an additional 25 page chapter on centroid computation for many of the distances presented. Instead, we will present a discussion on binary data as an example to show the basic principle of implementing clustering methods within our framework.

3.2 Binary Data

A special case of nominal data are m-dimensional binary observations where $\mathcal{X} = \{0, 1\}^m$. Obviously here the K-modes and K-median algorithms are exactly the same and produce binary centroids such that $\mathcal{C} = \mathcal{X}$. Most popular for clustering binary data is probably again the K-means algorithm where the centroids give the conditional marginal probabilities of observing a 1 given the cluster membership and $\mathcal{C} = [0, 1]^m$. Hence, K-means can be seen as fitting a finite binomial mixture model maximizing the *classification likelihood* (Fraley and Raftery, 2002).

In application domains like marketing the distances listed above are often not ideal because they treat zeros and ones symmetrically, see Section 6 for an example. Special distance measures for binary data have been used in statistics and cluster analysis for a long time (Anderberg,

1973), but mostly in combination with hierarchical cluster methods. For partitioning methods they are less common, see e.g., Leisch et al. (1998) for a hard competitive learning algorithm, or Strehl et al. (2000) for a comparison of K-means-type and graph-partitioning methods. Function `pam` described in Kaufman and Rousseeuw (1990) does partitioning around medoids for arbitrary distance measures, but in this case it needs the matrix of all pairwise distances between the observations as input, which is infeasible for larger data sets.

Consider two m-dimensional binary vectors $\mathbf{x}$ and $\mathbf{y}$. We define the 2×2 contingency table

		$\mathbf{x}$		
		1	0	
$\mathbf{y}$	1	α	β	$\alpha+\beta$
	0	γ	δ	$\gamma+\delta$
		$\alpha+\gamma$	$\beta+\delta$	m

where α counts the number of dimensions where both $\mathbf{x}$ and $\mathbf{y}$ are one, β the number of dimensions where $\mathbf{x}$ is zero and $\mathbf{y}$ is one, etc.. In the following we restrict ourselves to distance measures of type $d(\mathbf{x},\mathbf{y}) = d(\alpha,\beta,\gamma,\delta)$. E.g., the well known Hamming distance (number of different bits of $\mathbf{x}$ and $\mathbf{y}$) can be written as $d(\mathbf{x},\mathbf{y}) = \beta+\gamma$, which is also the absolute distance and squared Euclidean distance.

Asymmetric distance measures usually give more weight to common ones than common zeros, e.g., the Jaccard coefficient (e.g., Kaufman and Rousseeuw, 1990) is defined as

$$d(\mathbf{x},\mathbf{y}) = \frac{\beta+\gamma}{\alpha+\beta+\gamma}. \tag{1}$$

Consider the data are from "yes/no" questions and 1 encodes "yes". Since d does not depend on δ, questions where both subjects answered "no" are ignored; d is the percentage of disagreements in all answers where at least one subject answered "yes".

3.2.1 Canonical Centroids for Jaccard Distance

Computation of binary centroids with respect to the Jaccard distance is a potentially very slow combinatorial optimization problem. Temporarily using $\mathcal{C} = [0,1]^m$ is a simple way of overcoming this problem. Let

$$\begin{aligned}
\tilde{\alpha} &= \mathbf{x}'\mathbf{c}, \\
\tilde{\beta} &= (1-\mathbf{x})'\mathbf{c}, \\
\tilde{\gamma} &= \mathbf{x}'(1-\mathbf{c}), \\
\tilde{\delta} &= (1-\mathbf{x})'(1-\mathbf{c})
\end{aligned}$$

for $\mathbf{c} \in [0,1]^m$. Note that $(\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta})$ are identical to $(\alpha,\beta,\gamma,\delta)$ for binary $\mathbf{c}$. Plugging these values into Equation (1) gives a simple way of extending the binary Jaccard distance to real-valued centroids:

$$\tilde{d}(\mathbf{x},\mathbf{c}) = \frac{\tilde{\beta}+\tilde{\gamma}}{\tilde{\alpha}+\tilde{\beta}+\tilde{\gamma}} = 1 - \frac{\mathbf{x}'\mathbf{c}}{|\mathbf{x}|+|\mathbf{c}|-\mathbf{x}'\mathbf{c}} \tag{2}$$

(the term $(\mathbf{x}'\mathbf{c})/(|\mathbf{x}|+|\mathbf{c}|-\mathbf{x}'\mathbf{c})$ is sometimes called the Tanimoto coefficient of similarity). Taking first and second order partial derivatives we get

$$\begin{aligned}
\frac{\partial \tilde{d}}{\partial c_i} &= -\frac{x_i(|\mathbf{x}|+|\mathbf{c}|)-\mathbf{x}'\mathbf{c}}{(|\mathbf{x}|+|\mathbf{c}|-\mathbf{x}'\mathbf{c})^2}, \\
\frac{\partial^2 \tilde{d}}{\partial c_i^2} &= \frac{2(1-x_i)(x_i(|\mathbf{x}|+|\mathbf{c}|)-\mathbf{x}'\mathbf{c})}{(|\mathbf{x}|+|\mathbf{c}|-\mathbf{x}'\mathbf{c})^3}.
\end{aligned}$$

For $x_i \in \{0,1\}$ and $c_i \in [0,1]$ it can easily be seen that

$$0 \le \mathbf{x}'\mathbf{c} \le |\mathbf{c}|,$$

hence

$$\frac{\partial^2 \tilde{d}}{\partial c_i^2} \begin{cases} = 0, & x_i = 1 \\ \leq 0, & x_i = 0 \end{cases}$$

and $\tilde{d}$ is an element-wise non-convex function in $\mathbf{c}$ with no local minima inside the unit hypercube. Taking averages over the whole data set X_N does not change the sign of second derivatives, obviously the same is true for

$$\tilde{D}(X_N, \mathbf{c}) = \frac{1}{N} \sum_{n=1}^{N} \tilde{d}(\mathbf{x}_n, \mathbf{c}).$$

It follows directly that all minima of $\tilde{D}$ are on the boundaries of the unit hypercube, the minimizer is an element of $\{0, 1\}^m$, and minimizing $\tilde{D}$ on the unit hypercube gives the same centroids as directly minimizing the Jaccard distance. As $\tilde{D}$ is a continuous function in $\mathbf{c}$, any general purpose optimizer with box constraints can be used, and convergence is usually rather fast.

3.2.2 Expectation-based Centroids

From a practitioners point of view it is very attractive to obtain real-valued centroids that can be interpreted as probabilities of observing a 1, which corresponds to using the cluster-wise means as centroids. If $\mathbf{x}$ is a binary data vector and $\mathbf{c}$ contains the means of m independent binomial variables, then the expected values of the contingency table entries between $\mathbf{x}$ and a random independent vector $\mathbf{y}$ drawn from the multivariate binomial distribution with mean $\mathbf{c}$ are given by $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta})$ as defined above. Thus distance $\tilde{d}(\mathbf{x}, \mathbf{c})$ is an approximation of the expected average distance of $\mathbf{x}$ from another point in a cluster with mean vector $\mathbf{c}$.

Putting the pieces together results in a fast cluster algorithm for binary data that is very similar to standard K-means:

1. Start with a random set of initial centroids C_K (e.g., by drawing K points from X_N).
2. Assign each point $\mathbf{x}_n \in X_N$ to the cluster of the closest centroid with respect to $\tilde{d}$.
3. Use the cluster-wise means as new set of centroids.
4. Repeat steps 2 and 3 until convergence.

The only difference to K-means is to assign points to clusters using $\tilde{d}$ instead of Euclidean distance. Because the cluster-wise means are not the canonical centroids, convergence of the algorithm is not guaranteed, although in our experience this is not a problem in practice.

The expectation-based centroid method is of course applicable to all distances for binary data that can be written as $d(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta})$. If d is a linear function like for the Hamming distance, then $\tilde{d}$ is exactly the average expected distance of x to another point in the cluster.

4 Implementation

In order to make it easy for data analysts to try out a wide variety of distance measures on their data sets we have written an extensible implementation of the generalized K-means algorithm for the R statistical computing environment (R Development Core Team, 2005) using S4 classes and methods (Chambers, 1998). R package `flexclust` is freely available from the Comprehensive R Archive network at `http://cran.R-project.org`. The main function `kcca()` uses a family concept similar to the implementation of generalized linear models in S (Chambers and Hastie, 1992). In accordance to the abstract definition of the K-centroids cluster problem a KCCA family consists of the following 2 parts:

`dist`: A function taking N observations and K centroids as inputs and returning the $N \times K$ matrix of distances between all observations and centroids.

`cent`: An (optional) function computing the centroid for a given subset of the observations.

4.1 Example: K-Means

As a simple example consider the ordinary K-means algorithm. A function to compute Euclidean distance between observations and centroids is

```
distEuclidean <- function(x, centers)
{
    z <- matrix(0, nrow=nrow(x), ncol=nrow(centers))
    for(k in 1:nrow(centers)){
        z[,k] <- sqrt( colSums((t(x) - centers[k,])^2) )
    }
    z
}
```

For centroid computation we use the standard R function `colMeans()`. A KCCA family object can be created using

```
> kmFam <- kccaFamily(dist = distEuclidean, cent = colMeans)
```

and that is all that it takes to create a "brand new" clustering method for a given distance measure. It is not even necessary to write a function for centroid computation, if `cent` is not specified, a general purpose optimizer is used (at some speed and precision penalty). The partition in the upper left panel of Figure 1 was computed using

```
> x <- matrix(rnorm(1000), ncol = 2)
> cl1 <- kcca(x, 10, family = kmFam)
> cl1

kcca object of family 'distEuclidean'

call:
kcca(x = x, k = 10, family = kmFam)

cluster sizes:
 1  2  3  4  5  6  7  8  9 10
30 89 32 58 55 31 79 46 41 39
```

and finally plotted with the command `image(cl1)`.

Package `flexclust` is also flexible with respect to centroid computation. E.g., to get trimmed K-means (Cuesta-Albertos et al., 1997) as a robust version of the original algorithm, we can use the standard R function `mean()` (which allows trimming) instead of `colMeans()` for centroid computation. For convenience trimmed K-means is already implemented and can be used by setting the `trim` argument of `kccaFamily` to a positive value (the amount of trimming).

4.2 Example: Jaccard Distance

As a more advanced example consider we want to implement a clustering algorithm using the Jaccard distance, for both the canonical centroids as well as expectation-based centroids. Writing Equation 2 in vectorized S code gives

```
distJaccard <- function(x, centers)
{
  nx <- nrow(x)
  nc <- nrow(c)

  xc <- x %*% t(centers)
```

```
  denominator <-
    matrix(rowSums(x), nrow=nx, ncol=nc) +
    matrix(rowSums(centers), nrow=nx, ncol=nc, byrow=TRUE) - xc

  return(1 - xc/denominator)
}
```

We use R's general purpose optimization function `optim()` for centroid computation on the unit hypercube in

```
centOptim01 <- function(x, dist)
{
    foo <- function(p)
        sum(dist(x, matrix(p, nrow=1)))

    optim(colMeans(x), foo, lower=0, upper=1, method="L-BFGS-B")$par
}
```

such that we can define two new clustering families

```
> jaccFam <- kccaFamily(dist = distJaccard, cent = centOptim01)
> ejacFam <- kccaFamily(dist = distJaccard, cent = colMeans)
```

for canonical and expectation-based centroids, respectively. The R code listed above is all that is needed for family specification, actual usage is shown in the case study in Section 6 below.

4.3 CPU Usage Simulation

An obvious question is how large a speed penalty we pay for clustering using only interpreted code. We compare the function with two other popular R functions for partitioning cluster analysis: `kmeans` from package `stats` (R Development Core Team, 2005) and `clara` from package `cluster` (Struyf et al., 1997). We use 5-dimensional standard normal data sets of size $N = 1000, 5000, 10000, 20000$ and partition it into $K = 5, 10, 25, 50, 100$ clusters. Figure 2 shows the median CPU times in seconds on a Pentium M processor with 1.4GHz under Debian Linux after running each algorithm 5 times on each combination of N and K. Function `clara()` was run using $N/1000$ subsamples of size 1000 such that on average each data point is used in one subsample. Note that `clara()` can be made arbitrarily fast/slow by modifying these two parameters (with an obvious tradeoff on precision), the main purpose of this comparison is to show that clustering in interpreted code only is not prohibitively slow.

`kmeans()` clearly is much faster than the other two. Although `clara()` internally uses compiled C code, it is slower than `kcca()` for large number of clusters, probably due to the computationally more expensive exchange algorithm. Both `clara()` and `kcca()` return much more complex objects than `kmeans` containing auxiliary information for plotting and other methods. All three functions scale approximately linear with respect to memory and time for larger K and N, respectively.

The main reason that partitioning cluster analysis can be efficiently implemented in interpereted code is that function `distEuclidean()` has only a `for` loop over the number of centers, all other operations use vectorized R expressions which are actually evaluated in C code. We do pay a speed penalty, but only in the magnitude that users are already willing to pay for clustering algorithms like `clara()`.

4.4 Other Features of FlexClust

The `dist` slot of a KCCA family object can of course be used to assign arbitrary new data to clusters, such that a corresponding R function (in S speak: the `predict()` method for KCCA objects) comes for free. Instead of a distance measure the user may also specify a similarity measure. Function `stepKcca()` can be used to automatically restart the algorithm several times

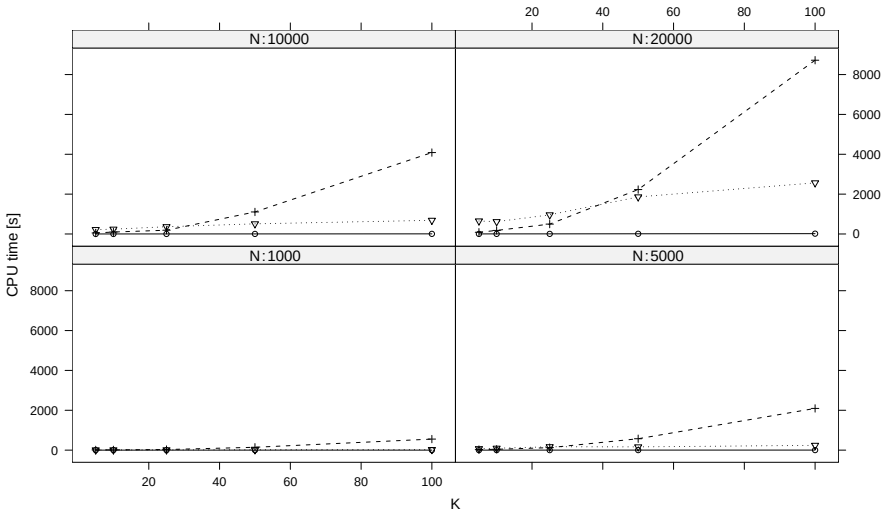


Figure 2: Median CPU times for R functions `kmeans()` (circles), `clara()` (crosses) and `kcca()` (triangles) on 5-dimensional standard normal data for varying K and N.

to avoid local minima and try out solutions for different numbers of clusters. We have also integrated the simulated annealing optimization by Hand and Krzanowski (2005).

A new interface to the functionality of `cclust` (Dimitriadou, 2005) provides faster parameter estimation in compiled code using the generalized K-means, learning vector quantization and neural gas algorithms (for Euclidean and Manhattan distance). It returns fully compatible KCCA objects including the corresponding families and auxiliary information for visualization. In addition there is a conversion function which transforms "partition" objects as returned by `pam()` to KCCA objects in case one wants to use the visualization features of `flexclust` on them.

Function `qtclust()` implements a generalized version of the QT cluster algorithm by Heyer et al. (1999) for the maximum radius problem. It also uses KCCA family objects and hence provides the QT algorithm for arbitrary distance measures without additional programming effort. Once an appropriate family object for a distance measure of interest is implemented, both the K-centroids problem and the maximum radius problem can be solved.

5 The Neighborborhood Graph

When a partition of potentially high-dimenstional data has been obtained, graphical representation of the cluster object can help to better understand the relationships between the segments and their relative position in the data space $\mathcal{X}$. Pison et al. (1999) project the data into two dimensions using principal component analysis or multi-dimensional scaling, draw a fully connected graph of all centroids and represent the segments by cluster-wise covariance ellipses or minimum spanning ellipses. Another obvious choice for projecting the data into 2-d would be discriminant coordinates (Seber, 1984) using cluster membership as grouping variable. Hennig (2004) introduces a collection of asymmetric projection methods which try to seperate one cluster as good as possible from all others, the clusters themselves are represented by the projections of the data points.

Whatever projection method is used, points that are close to each other in the 2-dimensional projection may have arbitrary distance in the original space. Several methods for visualizing the results of centroid-based cluster algorithms are based on the idea of using the centroids as nodes of a directed graph, see also Mazanec and Strasser (2000). Topology-representing networks (TRN, Martinetz and Schulten, 1994) use a competitive learning algorithm for parameter estimation and count how often a pair of centroids is closest / second-closest to a data point. Centroid pairs

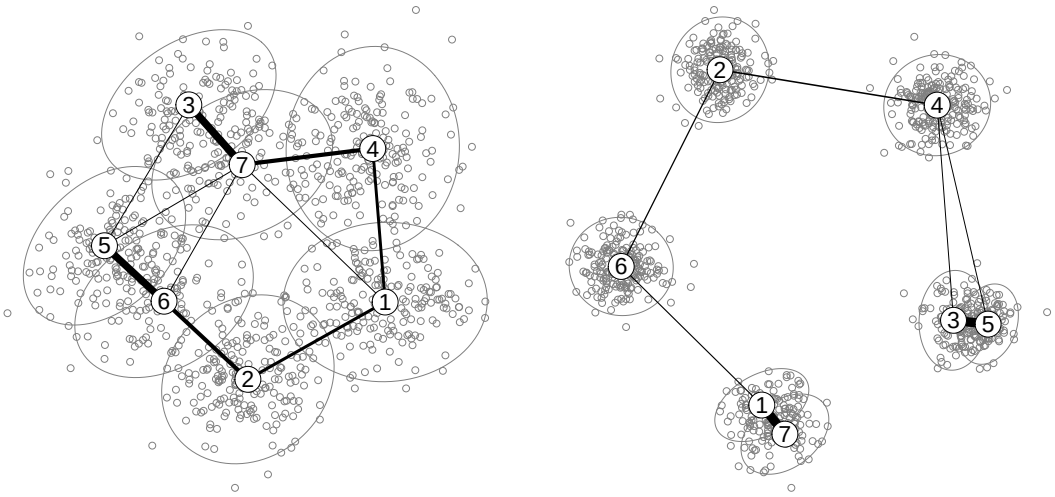


Figure 3: The neighborhood graphs for 7-cluster partitions of 5 Gaussians with good and bad separation together with 95% confidence ellipses for the clusters.

which have a positive count during the last iterations are connected. How many iterations are used for calculating the graph is a tuneable "lifetime" parameter of the algorithm. A disadvantage is that the resulting graphs convey no information on how well two clusters are separated from each other, because two centroids are either connected or not connected.

Combining TRN graphs with the main idea of silhouette plots (Rousseeuw, 1987) we propose to use mean relative distances as edge weights. Silhouettes measure how well each data point is clustered, while we need to measure how separated pairs of clusters are. Let $A_k \subset X_N$ be the set of all points in cluster k,

$$\tilde{c}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{c} \in C_K \setminus \{c(\mathbf{x})\}} d(x, \mathbf{c})$$

denote the second-closest centroid to $\mathbf{x}$ and let

$$A_{ij} = \{\mathbf{x}_n \,|\, \mathbf{c}_i = c(\mathbf{x}_n),\, \mathbf{c}_j = \tilde{c}(\mathbf{x}_n)\}$$

be the set of all points where $\mathbf{c}_i$ is the closest centroid and $\mathbf{c}_j$ is second-closest. Now we define edge weights

$$e_{ij} = \begin{cases} |A_i|^{-1} \sum_{x \in A_{ij}} \frac{2d(x,\mathbf{c}_i)}{d(x,\mathbf{c}_i)+d(x,\mathbf{c}_j)}, & A_{ij} \neq \emptyset \\ 0, & A_{ij} = \emptyset \end{cases}$$

If $e_{ij} > 0$, then at least one data point in segment i has $\mathbf{c}_j$ as second-closest centroid and segments i and j have a common border. If e_{ij} is close to 1, then many points are almost equidistant to $\mathbf{c}_i$ and $\mathbf{c}_j$ and the clusters are not separated very well. The graph with edge weights e_{ij} is a directed graph, to simplify matters we use the corresponding undirected graph with average values of e_{ij} and e_{ji} as edge weights in all figures of this paper.

The neighborhood graph is the `plot()` method for objects of class `"kcca"`. It has an optional `project` argument which can be used for higher-dimensional data to use (almost) arbitrary projections of the graph into 2-d.

Figure 3 shows neighborhood graphs for two data sets with 5 Gaussian clusters each. Both data sets are two-dimensional (hence no projection necessary) and have been clustered using K-means with 7 centers, the "wrong" number of clusters was intentional to show the effect on the graph. Both graphs show the ring-like structure of the data, the thickness of the lines is proportional to the edge weights and clearly shows how well the corresponding clusters are separated. Triangular

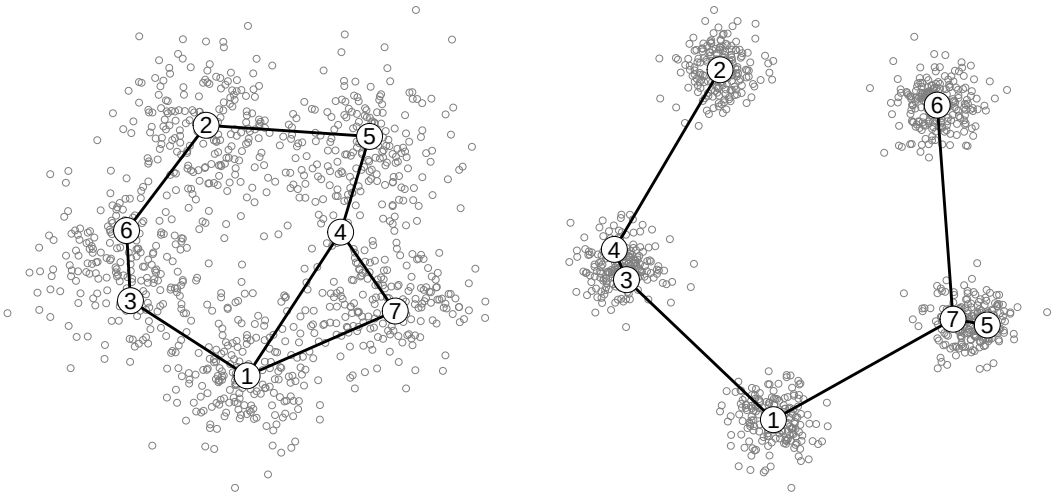


Figure 4: Traditional TRN graphs for the Gaussian data.

structures like in the left panel correspond to regions with almost uniform data density that are split into several clusters, see also Figure 1. Centroid pairs connected by one thick line and only thin lines to the rest of the graph like 1/7 and 3/5 in the right panel correspond to a well-separated data cluster that has wrongly been split into two clusters.

Traditional TRN graphs constructed as described in Martinetz and Schulten (1994) are shown in Figure 4. Centroids and connection graph have been computed using the TRN32 program by Mazanec (1997), the results have then been imported to `flexclust` for plotting only. The main difference is that in neighborhood graphs the line width conveys information about cluster separation, while in TRN graphs two centroids are either connected or not, so all lines have the same width. An additional advantage of neighborhood graphs is that they are computed off the fitted partition and hence can be combined with any centroid-based cluster algorithm, while the construction of TRN graphs is limited to one special algorithm. If the lifetime parameter of the TRN algorithm is not too large the connecting patterns of neighborhood graphs and TRN graphs tend to be very similar, thus neighborhood graphs can be seen as an extension of TRN graphs.

Figure 5 shows neighborhood graphs for Gaussian clusters located at the corners of 3- and 4-dimensionsional unit cubes. Both data sets have been projected to 2-d using principal component analysis. For the 3-dimensional data the cube structure can easily be inferred from the graph, while different colors and glyphs for the clusters can of course not visualize this structure. The graph for the 4-dimensional data looks like a big mess at first sight, but the true structure of the data can be seen when looking more closely. E.g., there is one 3-dimensional subcube looking like a prism formed by all centroids with odd numbers and another one formed by the even numbers (in this example cluster numbers are not random but were relabeled to make explaining the structure easier). Other prisms are formed by numbers 1–8 and 9–16. All centroids have 4 neighbors.

6 Case Study: Binary Survey Data

We now compare the K-means, K-medians and the two Jaccard-based methods for binary data from the Guest Survey Austria (GSA), a tri-annual national survey among tourists in Austria done by the National Tourism Organization. We use a subset of the data from the winter season of 1997 on 25 vacation activities. Winter tourism plays a major role not only within the Austrian tourism industry but for the Austrian economy as a whole, the main attraction being downhill skiing in the Alps.

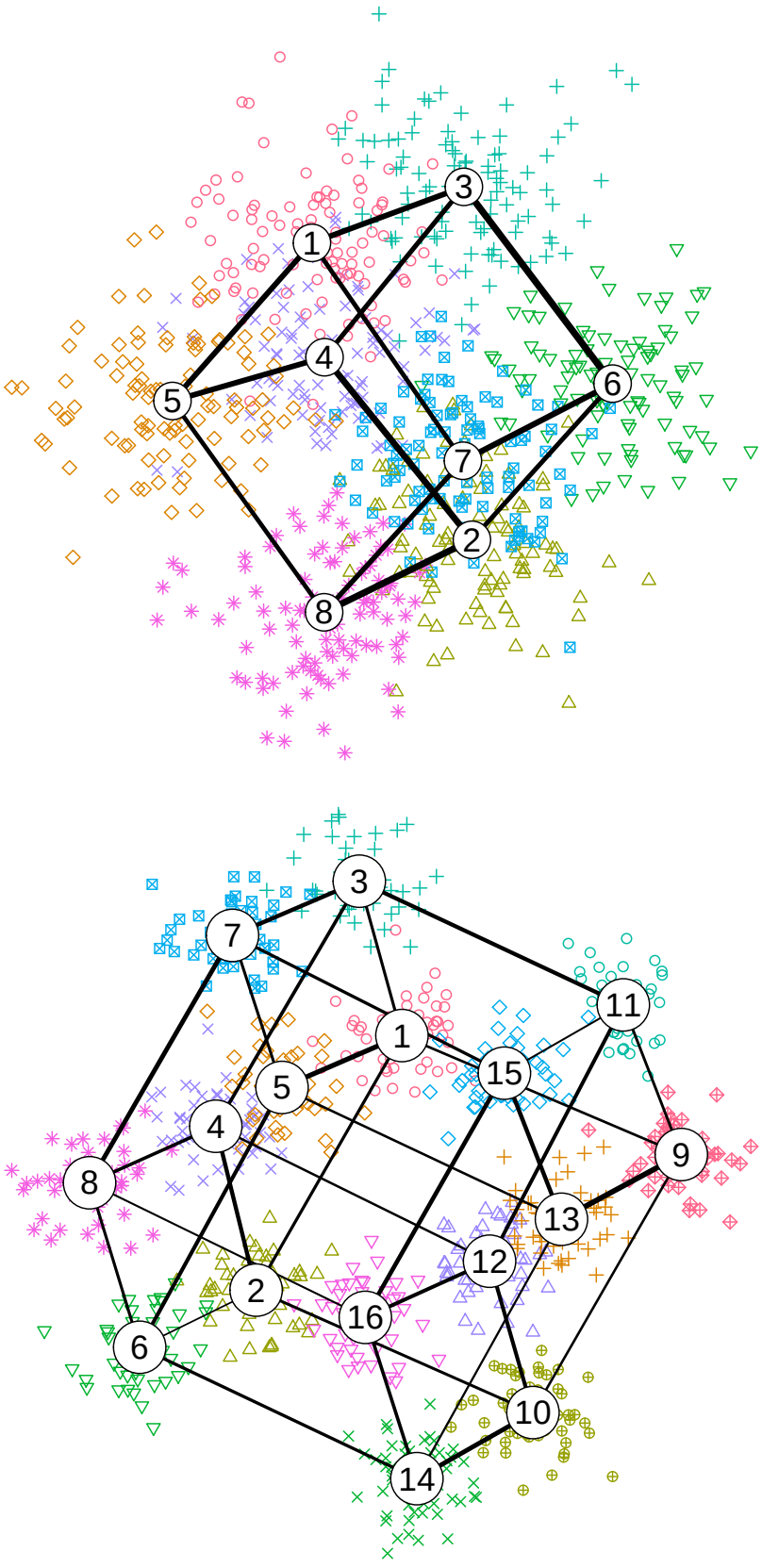


Figure 5: Neighborhood graphs for clusters at corners of unit cubes in 3 and 4 dimensions.

Each respondent was asked to state whether he or she undertook each one of the activities during the stay or not, totalling in 2958 respondents with complete answer profiles. The average participation in those activities ranged between 1% percent (horseback riding) and 75% (relaxing). In addition to this segmentation base, a number of descriptive pieces of information of demographic, socioeconomic, psychographic, attitudinal or behavioral nature were available for each respondent, see Dolničar and Leisch (2003) for a more detailed description and analysis of the data.

Data like this are prototypical for situations where the Jaccard distance makes more sense than Euclidean distance for interpretation: Two persons both going cross country skiing on their winter vacation clearly have more in common than two persons who both happen not to be interested in this activity. Previous analyses of this data set show that no clear density clusters exist, any segmentation will induce an artificial grouping on the data. Nevertheless it makes a lot of sense from a marketing researcher's point of view to cluster the data in order to partition the market into smaller subgroups of consumers which subsequently can be addressed by marketing actions tailored for the respective market segment. E.g., Mazanec et al. (1997) do not assume the existence of natural segments claiming that distinct segments rarely exist in empirical data sets and redefining market segmentation to be a construction task rather than a search mission for natural phenomena.

If matrix `gsa` contains the data a partition using expectation-based Jaccard distance is computed by

```
> gsa.cl <- kcca(gsa, k = 6, family = ejacFam)
```

The actual choice of expectation-based Jaccard with $K = 6$ clusters as shown in Figures 6 and 7 has been made manually by comparing various solutions and selecting the one which made most sense from the practitioners point of view. This may seem unsatisfying because the decision is subjective, but cluster analysis here is used as a tool for exploratory data analysis and offers simplified views of a complex data set. A principal component analysis of the data can be obtained by

```
> gsa.pca <- prcomp(gsa)
```

Combining both results we can plot the neighborhood graph of the partition in the space spanned by the first two principal components of the data using the command

```
> plot(gsa.cl, data = gsa, project = gsa.pca)
```

which is shown in Figure 6. Note that any projection object for which a `predict()` method exists can be used, not only return values of `prcomp()` like `gsa.pca`. The plot supports the previous finding of no clear cluster structure in the data, using other projection methods and distance measures gives similar pictures. There is a "path" of centroids given by sequence 6-2-5-1-4, cluster 3 is connected to 2 and 4 to lesser extent. This path is only present in the expectation-based Jaccard partition and was one reason for choosing this solution. The overlap of the projected clusters itself is not a direct indication for a lack of cluster separation, as it could be an artefact of the projection. However, from each node of the graph there is a path connecting it with the other nodes with large edge weights, and in addition the graph basically forms a loop, these are strong indications for no clear cluster structure.

Figure 7 shows the activity profiles of clusters 4 and 6, which are located at the opposite ends of the neighborhood graph. The barplots show the mean values of the clusters which equals the percentage of tourists undertaking the respective activity. The thin bars with the dots give the corresponding totals over the complete sample for comparison. Tourists in cluster 6 do three activities above average (alpine skiing, snowboard, disco), checking on background variables shows that the segment is significantly younger than the rest (no surprise given the activities). Profiles for clusters 2-5-1 and 3 are omitted for brevity. Clusters 2-5-1 are all skiers, with increasing amount of additional activities: cluster 2 prefers relaxing, walking and shopping to the disco, cluster 5 in addition also uses pool and sauna, and cluster 1 are tourists claiming to do almost every activity above average (an answer tendency often seen in questionaire data).

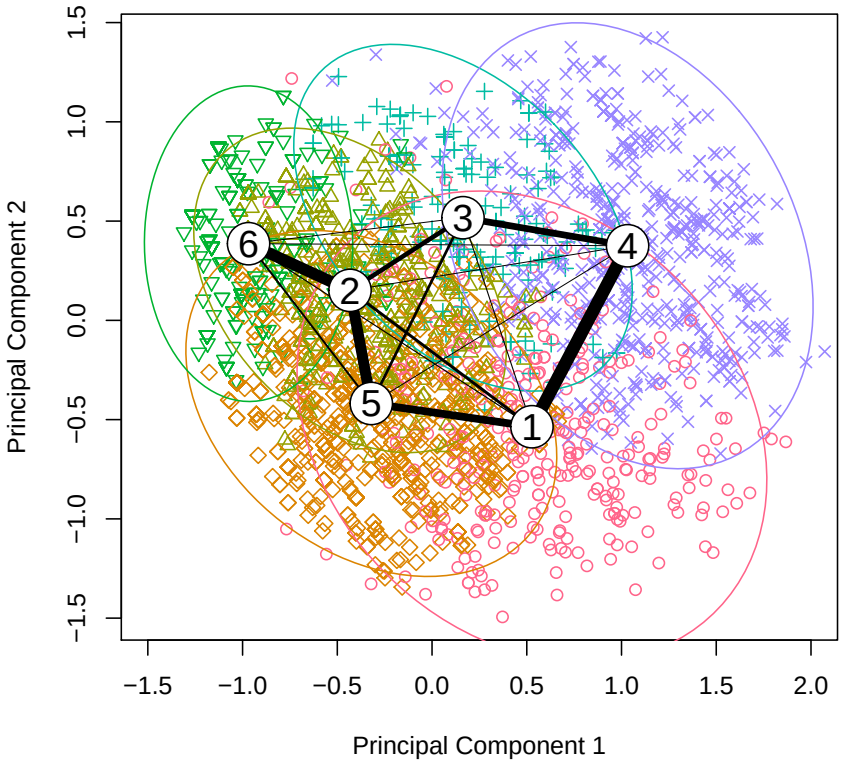


Figure 6: Projection of neighborhood graph of a Jaccard clustering of the GSA data on the first two principal components.

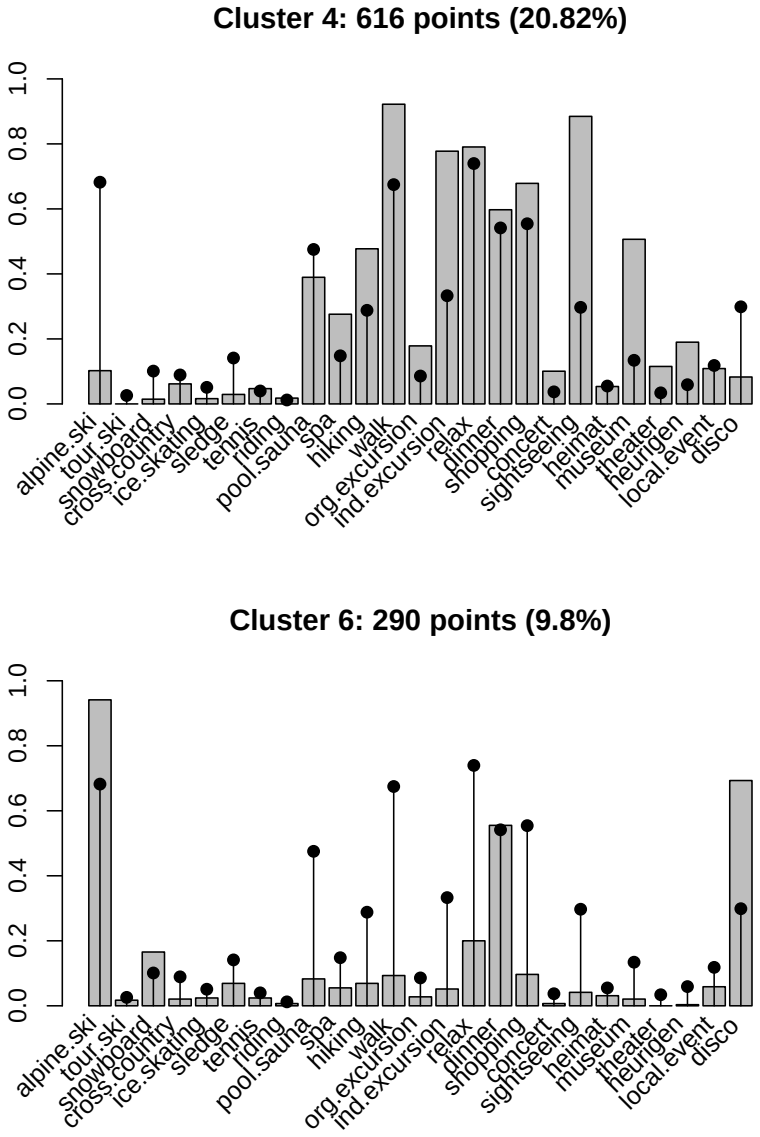


Figure 7: Two market segments from the expectation-based Jaccard partition of the GSA data: "Sightseers" (top) and "Young Skiers" (bottom).

Clusters 3 and 4 are the non-skiing winter tourists in Austria, the main difference is that cluster 4 (profile shown in Figure 7) contains typical city tourists with a lot of sightseeing and visits to museums, while cluster 3 is mainly interested in relaxing and hiking.

Because the cluster algorithm imposes a topology on this data set, the distance measure used is of uttermost importance. We evaluate the effect of distance measure on the partition result by

1. drawing bootstrap samples from the original data,
2. clustering the bootstrap samples into 6 clusters using each of the 4 algorithms, and
3. comparing the 4 partitions using the adjusted Rand index (Hubert and Arabie, 1985).

A Rand index of 1 corresponds to identical partitions and a Rand index of 0 corresponds to agreement by chance given cluster sizes. To make results more comparable we initialize each cluster method with either

- the K-means solution after 1 iteration, i.e., a random partition according to Euclidean distance, or
- the fully converged K-means solution.

We did the same simulations also for different numbers of clusters than 6, the results were very similar and are omitted here for simplicity.

Figure 8 shows boxplots of all pairwise comparisons for 100 bootstrap samples. For random initialization K-means and expectation-based Jaccard are most similar, which is not too surprising because both use the same method for centroid computation. Second in similarity are K-means and K-medians, the rest has average Rand values of 0.5. Even if we start in a fully converged K-means solution, all distance measures change the partition considerably by dropping to average Rand values of 0.75 compared with the initialization. Furthermore, they seem to drift in different directions from the initial values, because the similarity between K-medians and the Jaccard-based methods is lower than their respective agreements with K-means.

7 Summary and Future Work

The distance measure used for partitioning cluster analysis has major impact on the resulting data segments, yet both software and applications in many areas use only a few standard distance measures due to their computational efficiency. The usage of batch algorithms like generalized K-means enables researchers to easily explore a wide variety of distance measures with only minimal programming effort. Modern computers are fast enough that this flexibility with respect to distance measures is not prohibitive even in interpreted languages like R or S-PLUS for larger data sets with several ten thousands of observations.

If the data contain true density clusters and a parametric family for this density is known, or it is expected that the data density can be approximated by a parametric family, then the best method to recover those clusters is probably a finite mixture model (McLachlan and Peel, 2000). If no parametric family seems appropriate, then one should choose a distance or similarity measure that reflects the shape of the clusters that are expected to be found, which need not be circular (Euclidean distance) or square (Manhattan distance) in all cases.

If a partition is imposed on the data, like it is often done for market segmentation purposes or for finding groups of genes with similar expression pattern, then the distance controls the shapes of the segments. As the example in the case study has illustrated, different distance measures have of course different optima for the K-centroids cluster problem, and an optimal solution with respect to Euclidean distance need not even be a local optimum for other distances. In the case of no clear cluster structure there is no "correct" solution, clustering is exploratory data analysis and different methods present different views of the data. Getting more than one view can only increase our knowledge about the data and which partition to choose should be decided using background knowledge about the application at hand.

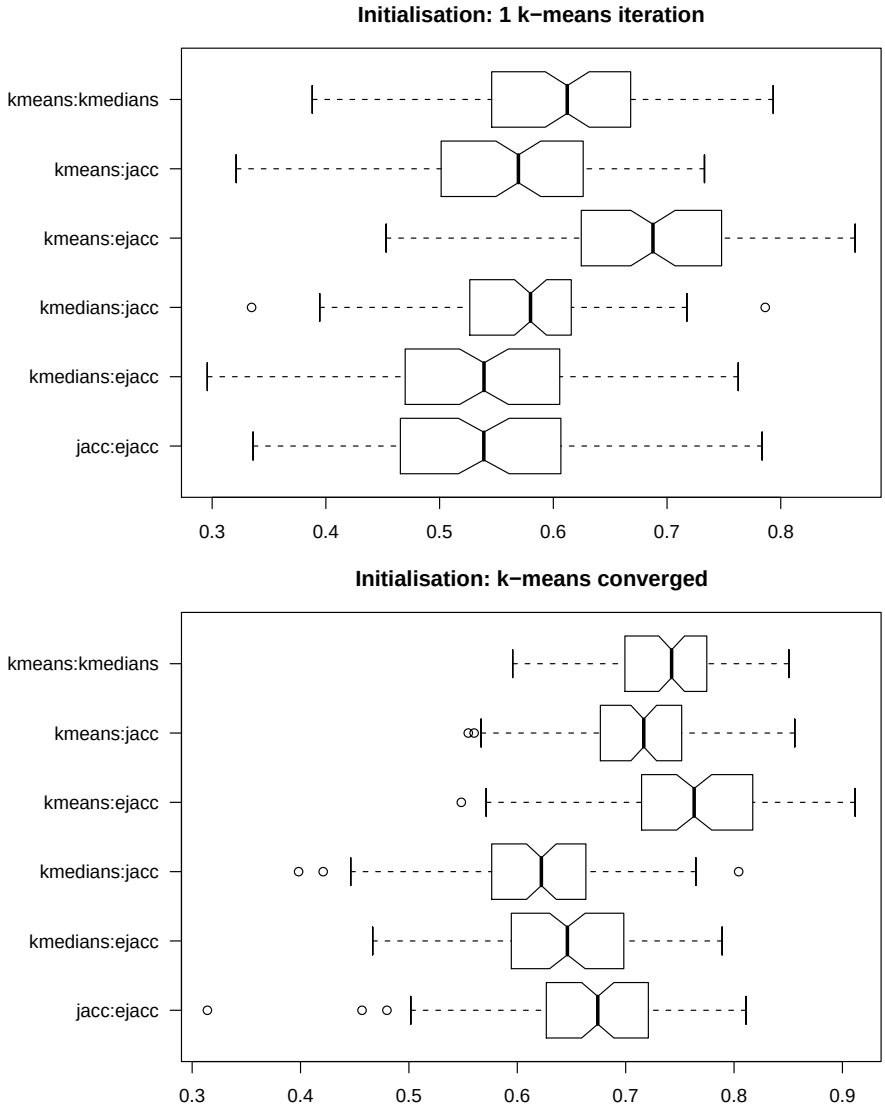


Figure 8: Rand index for pairwise comparison of cluster methods on 100 bootstrap samples of the GSA data.

The primary goal of `flexclust` is extensibility, it enables researchers to easily try out almost arbitrary distance measures and centroid computations for partitioning cluster analysis. E.g., better robustness against outliers can either be obtained by using more robust distance measures (L^1 instead of L^2, ...), using more robust centroid computations (trimming, ...) or a combination of both. The necessary programming effort is very limited, and because `kcca()` always returns proper KCCA objects, a growing collection of methods like the visualization tools used in this paper can be used for all these "new" clustering methods without any additional effort. There is no intent to replace packages implementing other cluster algorithms like `pam()` in package `cluster`, `flexclust` should rather be seen as a complement to those.

The simulated Rand indices for the binary data in the case study show that some clustering methods are more similar than others, a pattern that can also be observed for other data sets. It would be interesting to make a comprehensive benchmark study investigating how various distance measures compare to each other. Current research projects of our group investigate this question for the special cases of clustering market basket data and clustering gene expression profiles over time.

In the future we want to address the extensibility of online methods like exchange algorithms. Currently those need an implementation in compiled code to run in reasonable time for larger data sets, this will certainly change over the next decade. In the meantime a little bit of C or FORTRAN programming will be required, but systems like R can load compiled code at runtime, such that an extensible implementation should be possible.

Visualization for non-ellipsoid clusters is also a current research issue. Linear projections from high-dimensional spaces into 2-d are often ellipsoid even for highly non-Gaussian data due to the central limit theorem, but this is not always the case. The neighborhood graph helps to visualize relations between adjacent clusters for arbitrary distance measures, and its easy combination with a wide variety of projection methods allows again to get many different views of the data.

Acknowledgements

The author would like to thank Bettina Grün for valuable feedback and software debugging, and three anonymous referees for constructive comments and suggestions. This research was supported by the Austrian Science Foundation (FWF) under grant P17382-N12.

References

Anderberg, M. R., 1973. Cluster analysis for applications. Academic Press Inc., New York, USA.

Becker, R. A., Chambers, J. M., Wilks, A. R., 1988. The New S Language. Chapman & Hall, London, UK.

Chambers, J. M., 1998. Programming with data: A guide to the S language. Springer Verlag, Berlin, Germany.

Chambers, J. M., Hastie, T. J., 1992. Statistical Models in S. Chapman & Hall, London, UK.

Chaturvedi, A., Green, P. E., Carroll, J. D., 2001. K-modes clustering. Journal of Classification 18, 35–55.

Cuesta-Albertos, J. A., Gordaliza, A., Matran, C., 1997. Trimmed k-means: An attempt to robustify quantizers. The Annals of Statistics 25 (2), 553–576.

Dimitriadou, E., 2005. cclust: Convex Clustering Methods and Clustering Indexes. R package version 0.6-12.

Dolničar, S., Leisch, F., 2003. Winter tourist segments in Austria: Identifying stable vacation styles using bagged clustering techniques. Journal of Travel Research 41 (3), 281–292.

Everitt, B. S., Landau, S., Leese, M., 2001. Cluster Analysis, 4th Edition. Arnold, London, UK.

Fraley, C., Raftery, A. E., 2002. Model-based clustering, discriminant analysis and density estimation. Journal of the American Statistical Association 97, 611–631.

Hand, D. J., Krzanowski, W. J., 2005. Optimising k-means clustering results with standard software packages. Computational Statistics and Data Analysis 49, 969–973.

Hartigan, J. A., Wong, M. A., 1979. Algorithm AS136: A k-means clustering algorithm. Applied Statistics 28 (1), 100–108.

Hennig, C., 2004. Asymmetric linear dimension reduction for classification. Journal of Computational and Graphical Statistics 13 (4), 1–17.

Heyer, L. J., Kruglyak, S., Yooseph, S., 1999. Exploring expression data: Identification and analysis of coexpressed genes. Genome Research 9, 1106–1115.

Hubert, L., Arabie, P., 1985. Comparing partitions. Journal of Classification 2, 193–218.

Kaufman, L., Rousseeuw, P. J., 1990. Finding Groups in Data. John Wiley & Sons, Inc., New York, USA.

Kohonen, T., 1984. Self-organization and Associative Memory. Springer Verlag, New York, USA.

Kruskal, J., 1977. The relationship between multidimensional scaling and clustering. In: Ryzin, J. V. (Ed.), Classification and Clustering. Academic Press, Inc., New York, pp. 17–44.

Leisch, F., Weingessel, A., Dimitriadou, E., Sep. 1998. Competitive learning for binary valued data. In: Niklasson, L., Bodén, M., Ziemke, T. (Eds.), Proceedings of the 8th International Conference on Artificial Neural Networks (ICANN 98). Vol. 2. Springer, Skövde, Sweden, pp. 779–784.

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. In: Cam, L. M. L., Neyman, J. (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, Berkeley, CA, USA, pp. 281–297.

Martinetz, T., Schulten, K., 1994. Topology representing networks. Neural Networks 7 (3), 507–522.

Mazanec, J., 1997. TRN32: Topology Representing Network. Wirtschaftsuniversität Wien, Austria, extended Version 1.0 beta (32 bit).

Mazanec, J., Grabler, K., Maier, G., 1997. International City Tourism: Analysis and Strategy. Pinter/Cassel.

Mazanec, J. A., Strasser, H., 2000. A nonparametric approach to perceptions-based marketing: Foundations. Interdisciplinary Studies in Economics and Management. Springer Verlag, Berlin, Germany.

McLachlan, G., Peel, D., 2000. Finite Mixture Models. John Wiley and Sons Inc.

Pison, G., Struyf, A., Rousseeuw, P. J., 1999. Displaying a clustering with CLUSPLOT. Computational Statistics and Data Analysis 30, 381–392.

R Development Core Team, 2005. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Ramon, J., 2002. Clustering and instance based learning in first order logic. Ph.D. thesis, Department of Computer Science, K.U. Leuven, Belgium.

Ripley, B. D., 1996. Pattern recognition and neural networks. Cambridge University Press, Cambridge, UK.

Rousseeuw, P. J., 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20, 53–65.

Seber, G. A. F., 1984. Multivariate Observations. John Wiley & Sons, New York, USA.

Smet, F. D., Mathys, J., Marchal, K., Thijs, G., Moor, B. D., Moreau, Y., 2002. Adaptive quality-based clustering of gene expression profiles. Bioinformatics 18 (5), 735–746.

Strehl, A., Gosh, J., Mooney, R., 2000. Impact of similarity measures on web-page clustering. In: Proceedings of AAAI-2000: Workshop of Artificial Intelligence for Web Search. American Association for Artificial Intelligence, pp. 58–64.

Struyf, A., Hubert, M., Rousseeuw, P. J., 1997. Integrating robust clustering techniques in S-PLUS. Computational Statistics and Data Analysis 26, 17–37.

Venables, W. N., Ripley, B. D., 2000. S Programming. Springer.