



Spherical k -Means Clustering

Kurt Hornik

WU Wirtschaftsuniversität Wien

Ingo Feinerer

Technische Universität Wien

Martin Kober

WU Wirtschaftsuniversität Wien

Christian Buchta

WU Wirtschaftsuniversität Wien

Abstract

Clustering text documents is a fundamental task in modern data analysis, requiring approaches which perform well both in terms of solution quality and computational efficiency. Spherical k -means clustering is one approach to address both issues, employing cosine dissimilarities to perform prototype-based partitioning of term weight representations of the documents.

This paper presents the theory underlying the standard spherical k -means problem and suitable extensions, and introduces the R extension package **skmeans** which provides a computational environment for spherical k -means clustering featuring several solvers: a fixed-point and genetic algorithm, and interfaces to two external solvers (CLUTO and Gmeans). Performance of these solvers is investigated by means of a large scale benchmark experiment.

Keywords: spherical, clustering, text mining, cosine dissimilarity, R.

1. Introduction

Processing and analyzing textual data is one of the major challenges in modern data analysis. One key task is clustering text documents. Popular approaches can be grouped into the following three categories.

First, one can introduce suitable dissimilarity measures between the texts, and perform clustering based on these dissimilarities. String kernels (Lodhi, Saunders, Shawe-Taylor, Cristianini, and Watkins 2002) have become very popular in this context, as the corresponding dissimilarities can be computed rather efficiently, and be deployed in the context of modern kernel-based learning techniques such as kernel k -means clustering (Karatzoglou and Feinerer 2010).

Second, one can perform model-based clustering using probabilistic models for the generation of the texts, such as topic models (and variants thereof) for uncovering the latent semantic structure of a document collection based on a hierarchical Bayesian analysis of the texts (e.g., Griffiths, Steyvers, and Tenenbaum 2007; Blei and Lafferty 2009).

Finally, one can use a suitable “vector space model” (also known as “bag of words”) representation of the corpus (the collection of text documents), and use more traditional approaches for clustering multivariate numeric data. This model was first used in the SMART Information Retrieval System and represents documents (and queries) by vectors of term weights, and hence the corpus as a document-term weight matrix, with the so-called tf-idf (term frequency-inverse document frequency) scheme the most popular (Salton, Wong, and Yang 1975; Manning, Raghavan, and Schütze 2008). Given such a representation of documents d_i by term weight “feature vectors” x_i , one obvious approach to partitioning the documents into a given number k of groups is via minimizing a criterion function of the form

$$\sum_i d(x_i, p_{c(i)}),$$

over all assignments c of objects i to cluster ids $c(i) \in \{1, \dots, k\}$ and over all prototypes (often called centroids) p_1, \dots, p_k in the same feature space as the x_i , for a suitable dissimilarity measure d , i.e., by performing *k-prototypes partitioning with dissimilarity d* . It may seem straightforward to employ Euclidean dissimilarity (squared Euclidean distance) $d(x, p) = \|x - p\|^2$ and thus use standard k -means clustering. However, this typically over-represents long documents (with large aggregate term weights). To mitigate the effect of differing document lengths, Dhillon and Modha (2001) suggest to use the Euclidean dissimilarities of the projections of the feature vectors onto the unit sphere, or equivalently, employ the so-called *cosine dissimilarity*

$$d(x, p) = 1 - \cos(x, p) = 1 - \frac{\langle x, p \rangle}{\|x\| \|p\|}$$

based on the angle between the vectors. In fact, using the angles between the term weight vectors representing the documents and queries has long been the preferred method in information retrieval to compute relevance rankings of documents in a keyword search. Zhao and Karypis (2004) study seven different criterion functions for partitional document clustering via a comprehensive experimental evaluation involving 15 different datasets, and find very good performance for a criterion function (\mathcal{I}_2) corresponding to k -prototypes partitioning with cosine dissimilarity. Of course, this partitioning approach can be employed in arbitrary application domains where cosine dissimilarity is judged an appropriate measure of dissimilarity in the feature space for the measurements on the objects to be partitioned, e.g., for gene expression clustering (D’haeseleer 2005) or market basket analysis (Tan, Kumar, and Srivastava 2004).

The “spherical k -means algorithm” of Dhillon and Modha (2001) is a simple fixed-point heuristic for minimizing $\sum_i (1 - \cos(x_i, p_{c(i)}))$ which iterates between computing optimal cluster ids for fixed prototypes and computing optimal prototypes for fixed cluster ids. Banerjee, Dhillon, Ghosh, and Sra (2005) show that this algorithm is also obtained as an EM variant for Maximum Likelihood Estimation of the mean direction parameters of a uniform mixture of von Mises-Fisher (or Langevin) distributions with common concentration parameter κ , using hard-max classification E-steps (either directly or by letting $\kappa \rightarrow \infty$). This fixed-point algorithm typically converges very rapidly, and can easily be implemented from scratch or taking

advantage of available flexible prototype-based partitioning infrastructure as provided, e.g., in the extension packages **clue** (Hornik 2005) or **flexclust** (Leisch 2006) for R (R Development Core Team 2012). However, the algorithm may not converge to the optimum of the underlying criterion function, and different heuristics (e.g., based on local improvement strategies or different frameworks for solving hard optimization problems, such as genetic algorithms) may be able to find “better” partitions (with a smaller criterion value). In addition, heuristics for solving the spherical k -means optimization problem should be able to handle large corpora, resulting in large but typically very sparse corresponding document-term matrices, with reasonable efficiency by employing suitable sparse matrix representations and computations.

In this paper we discuss the R extension package **skmeans**, available from the Comprehensive R Archive Network at <http://CRAN.R-project.org/package=skmeans>, which provides a computational environment for spherical k -means clustering. It features solvers implementing the fixed-point (possibly enhanced by a local improvement strategy) and a genetic algorithm as well as interfaces to two standalone solvers (CLUTO and Gmeans). Available solvers can handle the most common sparse matrix formats for representing document-term matrices of large corpora, and are easily extended to handle additional formats.

This paper is organized as follows. Section 2 presents the theory of the (standard) spherical k -means clustering problem and extensions thereof. Section 3 discusses solution heuristics and their availability and implementation in package **skmeans**, and provides an example illustrating how to perform document clustering applications using the package. A large scale benchmark experiment analyzing the performance and efficiency of the available solvers is given in Section 4. Section 5 concludes the paper.

2. Theory

2.1. The standard spherical k -means problem

To fix terminology, the *standard spherical k -means problem* is to minimize

$$\sum_i (1 - \cos(x_i, p_{c(i)}))$$

over all assignments c of objects i to cluster ids $c(i) \in \{1, \dots, k\}$ and over all prototypes p_1, \dots, p_k in the same feature space as the feature vectors x_i representing the objects. (In text clustering, the objects are the documents to be partitioned.) Let n be the number of objects and $X = [x'_1, \dots, x'_n]'$ and $P = [p'_1, \dots, p'_k]'$ the data and prototype matrices with the feature vectors x_i and prototype vectors p_j as their rows, respectively, where $'$ denotes transpose. With the memberships μ_{ij} of objects i to classes j defined by

$$\mu_{ij} = \begin{cases} 1, & \text{if } c(i) = j \\ 0, & \text{otherwise,} \end{cases}$$

and the membership matrix $M = [\mu_{ij}]$, the standard spherical k -means problem can equivalently be formulated as minimizing

$$\Phi(M, P) = \sum_{i,j} \mu_{ij} (1 - \cos(x_i, p_j))$$

over all binary membership matrices M with unit row sums (i.e., $\mu_{ij} \in \{0, 1\}$ for all i and j and $\sum_j \mu_{ij} = 1$ for all i) and prototype matrices P .

For fixed prototypes P , Φ is obviously minimized over M by classifying objects to their nearest prototypes, i.e., choosing $c(i)$ as one j attaining $\min_j(1 - \cos(x_i, p_j))$.

For fixed memberships M , Φ is minimized over P if for each prototype p_j , $\sum_i \mu_{ij}(1 - \cos(x_i, p_j))$ is minimized. Now

$$\sum_i \mu_{ij}(1 - \cos(x_i, p_j)) = \sum_i \mu_{ij} \left(1 - \frac{\langle x_i, p_j \rangle}{\|x_i\| \|p_j\|} \right) = \sum_i \mu_{ij} - \left\langle \sum_i \mu_{ij} \frac{x_i}{\|x_i\|}, \frac{p_j}{\|p_j\|} \right\rangle,$$

which by the well-known Cauchy-Schwartz inequality is minimized iff

$$p_j \propto s_j(M) = \sum_i \mu_{ij} \frac{x_i}{\|x_i\|}. \quad (1)$$

In particular, if the feature vectors x_i are normalized to unit length $\|x_i\| = 1$, then optimal prototypes can be obtained as

$$p_j \propto \sum_i \mu_{ij} x_i = \sum_{i:\mu_{ij}=1} x_i = \sum_{i:c(i)=j} x_i,$$

i.e., as proportional to the sum over the feature vectors x_i of objects i in class j .

A fixed-point algorithm iterating between computing optimal M for fixed P and optimal P for fixed M will obviously not increase the $\Phi(M, P)$ values, which will thus (as non-increasing and trivially bounded below) converge to a limiting value. As pointed out by [Dhillon and Modha \(2001\)](#), this does not imply that the sequences of memberships or prototypes converge (by compactness, one can always choose convergent sub-sequences). In any case, there is no guarantee that the limiting Φ value is optimal.

Given a partition and a criterion function measuring its quality, one can always ask whether the value of the criterion function can be improved by moving a single object from one class to another (or performing a chain of such moves). In general, the corresponding computations may be prohibitively expensive. For the standard spherical k -means problem, [Dhillon, Guan, and Kogan \(2002\)](#) note that the effect of moving the i -th object from its cluster $j = c(i)$ to a different cluster l can be obtained as follows. Let $\Psi(M) = \min_P \Phi(M, P)$ be the minimal criterion value for fixed M . Then, using Equation 1,

$$\Psi(M) = \sum_j \left(\sum_i \mu_{ij} - \left\langle s_j(M), \frac{s_j(M)}{\|s_j(M)\|} \right\rangle \right) = n - \sum_j \|s_j(M)\|.$$

The move changes $s_j = s_j(M)$ to $s_j - \tilde{x}_i$ and $s_l = s_l(M)$ to $s_l + \tilde{x}_i$, where $\tilde{x}_i = x_i / \|x_i\|$, with criterion change $(\|s_j\| - \|s_j - \tilde{x}_i\|) + (\|s_l\| - \|s_l + \tilde{x}_i\|)$ and the new squared norms given by $\|s_j - \tilde{x}_i\|^2 = \|s_j\|^2 - 2s_j' \tilde{x}_i + 1$ and similarly $\|s_l + \tilde{x}_i\|^2 = \|s_l\|^2 + 2s_l' \tilde{x}_i + 1$. Thus, optimal single object moves (so-called first variation moves) can be computed in $O(n+k)$ steps provided that all object-prototype dissimilarities are available (these need to be computed for a fixed-point iteration anyway). In particular, one can attempt to improve the fixed-point algorithm via performing Kernighan-Lin style fixed-length chains of optimal single object moves to possibly improve the criterion ‘‘sufficiently enough’’ once the fixed-point iterations failed to do so, and in case of success resume the latter ([Dhillon et al. 2002](#)).

2.2. The extended spherical k -means problem

The criterion function for spherical k -means clustering can easily be generalized by replacing the μ_{ij} by μ_{ij}^m for exponents $m \geq 1$, as was done for the standard k -means problem to give the fuzzy k -means problem (Bezdek 1981). With $m > 1$, in general soft (fuzzy) partitions are obtained, with softness increasing with m . We are not aware of references suggesting this obvious generalization, which seems somewhat striking given the popularity of fuzzy clustering algorithms. Another seemingly obvious generalization is introducing object (“case”) weights w_i . This is particularly useful when trying to assess the stability of partitions with respect to changes in the data (distribution), which can conveniently be accomplished by varying the object weights.

Taking both generalizations into account, we obtain the *extended spherical k -means problem*, defined as the minimization of the criterion function

$$\Phi(M, P) = \sum_{i,j} w_i \mu_{ij}^m (1 - \cos(x_i, p_j))$$

over all (possibly soft) memberships matrices M with non-negative entries and unit row sums (i.e., $\mu_{ij} \geq 0$ for all i and j and $\sum_j \mu_{ij} = 1$ for all i , or equivalently, M is a stochastic matrix) and prototype matrices P .

For fixed prototypes P , Φ is minimized over M by choosing, for each object i , the memberships μ_{ij} such that $\sum_j w_i \mu_{ij}^m \delta_{ij}$ is minimal, where $\delta_{ij} = 1 - \cos(x_i, p_j)$ for notational convenience. Assume positive object weights. If $m = 1$, classification to the closest prototype is still optimal. Let $m > 1$. If some, say l , δ_{ij} are zero, optimality can be achieved by taking $\mu_{ij} = 1/l$ if $\delta_{ij} = 0$, and zero otherwise (if $l > 1$, other non-symmetric schemes are possible). Otherwise, we can obtain the optimal memberships using the Lagrange multiplier technique. The Lagrange function is $L(M, \lambda) = -\sum_{i,j} w_i \mu_{ij}^m (1 - \cos(x_i, p_j)) + \sum_i \lambda_i (\sum_j \mu_{ij} - 1)$, and setting the partial derivatives with respect to the μ_{ij} and λ_i to zero yields

$$\mu_{ij} = \frac{1/(w_i \delta_{ij})^{1/(m-1)}}{\sum_l 1/(w_i \delta_{il})^{1/(m-1)}}, \quad \delta_{ij} = 1 - \cos(x_i, p_j). \quad (2)$$

For fixed memberships M , Φ is minimized over P if for each prototype p_j , $\sum_i w_i \mu_{ij}^m (1 - \cos(x_i, p_j))$ is minimized. Now

$$\begin{aligned} \sum_i w_i \mu_{ij}^m (1 - \cos(x_i, p_j)) &= \sum_i w_i \mu_{ij}^m \left(1 - \frac{\langle x_i, p_j \rangle}{\|x_i\| \|p_j\|} \right) \\ &= \sum_i w_i \mu_{ij}^m - \left\langle \sum_i w_i \mu_{ij}^m \frac{x_i}{\|x_i\|}, \frac{p_j}{\|p_j\|} \right\rangle, \end{aligned}$$

which again by Cauchy-Schwartz is minimized iff

$$p_j \propto s_j(M) = \sum_i w_i \mu_{ij}^m \frac{x_i}{\|x_i\|}.$$

If $m = 1$, this reduces to the weighted sum $\sum_{i:c(i)=j} w_i x_i / \|x_i\|$ of the normalized feature vectors in class j .

From the above, one can readily obtain a fixed-point algorithm for the extended spherical k -means problem which again iterates between computing optimal M for fixed P and optimal P for fixed M . Again, the corresponding $\Phi(M, P)$ will converge to a limiting value, but not necessarily the optimal one. Note that $\Psi(M)$, the minimal criterion value for fixed M , is

$$\begin{aligned}\Psi(M) &= \sum_j \left(\sum_i w_i \mu_{ij}^m - \left\langle s_j(M), \frac{s_j(M)}{\|s_j(M)\|} \right\rangle \right) \\ &= \sum_{i,j} w_i \mu_{ij}^m - \sum_j \|s_j(M)\|.\end{aligned}$$

If $m = 1$ (but the weights are not necessarily all one), the first term is $\sum_{i,j} w_i \mu_{ij} = \sum_i w_i$ and hence constant, and the results of [Dhillon *et al.* \(2002\)](#) for optimal single object moves can straightforwardly be generalized, with $\tilde{x}_i = w_i x_i / \|x_i\|$ and new squared norms given by $\|s_j - \tilde{x}_i\|^2 = \|s_j\|^2 - 2s_j' \tilde{x}_i + w_i^2$ and similarly $\|s_l + \tilde{x}_i\|^2 = \|s_l\|^2 + 2s_l' \tilde{x}_i + w_i^2$. The situation is fundamentally different if $m > 1$ (where memberships may vary continuously). Suppose M is a fixed point. Consider an arbitrary smooth curve of memberships $t \mapsto M(t) = [\mu_{ij}(t)]$ passing through M at $t = 0$ (i.e., satisfying $M(0) = M$). The derivative of the criterion value of $M(t)$ optimized over P is

$$\begin{aligned}\frac{d\Psi(M(t))}{dt} &= \sum_{i,j} m w_i \mu_{ij}(t)^{m-1} \dot{\mu}_{ij}(t) - \sum_j \left\langle \frac{ds_j(M(t))}{dt}, \frac{s_j(M(t))}{\|s_j(M(t))\|} \right\rangle \\ &= \sum_{i,j} m w_i \mu_{ij}(t)^{m-1} \dot{\mu}_{ij}(t) \left(1 - \left\langle \frac{x_i}{\|x_i\|}, \frac{s_j(M(t))}{\|s_j(M(t))\|} \right\rangle \right)\end{aligned}$$

where $\dot{\mu}_{ij}(t) = d\mu_{ij}(t)/dt$. As all $M(t)$ are memberships, $\sum_j \mu_{ij}(t) = 1$ and hence $\sum_j \dot{\mu}_{ij}(t) = 0$. As M is a fixed point, we know from Equation 2 that $\mu_{ij}^{m-1}(1 - \cos(x_i, s_j(M)))$ does not depend on j . Hence, writing γ_i for the common value, $d\Psi(M(t))/dt|_{t=0} = \sum_{i,j} m w_i \gamma_i \dot{\mu}_{ij}(0) = \sum_i m w_i \gamma_i \sum_j \dot{\mu}_{ij}(0) = 0$. Thus, first order local improvements are not possible. In fact, we are not aware of references discussing direct local improvement heuristics for soft partitioning problems. The strategy of [Belacel, Hansen, and Mladenovic \(2002\)](#) (for the fuzzy k -means problem) employs restarts based on replacing a single prototype by a feature vector (resulting in non-local changes to the memberships).

3. Software

3.1. Interface

Using package `skmeans`, spherical k -means clustering is performed by function `skmeans` with synopsis

```
skmeans(x, k, method = NULL, m = 1, weights = 1, control = list())
```

with arguments `x`, `k`, `m` and `weights` giving the data matrix X , the desired number of groups k , the softness parameter m , and the object weights w , respectively, and `control` a list of suitable control parameters.

Argument `method` specifies the method employed for “solving” the corresponding (standard or extended) spherical k -means problem. Note that these are not *exact* solvers: following common optimization terminology, a solver is an algorithm/code attempting to minimize the criterion function, and a “solution” is what a solver delivers, which is not even guaranteed to deliver a value close to the true minimum (see Section 4 for a performance comparison of the available solvers on a collection of benchmark data sets). Argument `method` can be a character string specifying one of the built-in solvers (a fixed-point and a genetic algorithm, and interfaces to two standalone solvers (CLUTO and Gmeans) and the C code implementing the k -mean-directions algorithm of Maitra and Ramler (2010)) which are discussed in detail below, a function to be taken as a user-defined method, or NULL (default). By default, the genetic algorithm is used for obtained hard partitions, and the fixed-point algorithm otherwise. The fixed-point and genetic solvers allow to specify starting partitions via control parameter `start` in the form of character vectors naming built-in initialization methods, or lists of prototype matrices or `skmeans` results objects. This makes it possible to experiment with different initialization strategies, or attempt to combine methods for possible performance improvements (e.g., applying the local improvement heuristics to the results of CLUTO). By default, random sets of objects are taken as initial prototypes.

All methods return an object inheriting from classes "`skmeans`" and "`pclus`", the latter provided by package `clue` for representing the results of hard or soft prototype-based partitioning algorithms. These objects are lists with components including the following:

prototypes: The prototype matrix P .

membership: The membership matrix M (only provided if $m > 1$).

cluster: The class ids $c(1), \dots, c(n)$ of the closest hard partition (the partition itself if $m = 1$).

value: The value $\Phi(M, P)$ of the criterion function.

This allows to use the high-level functionality of `clue` for further computations on the results, in particular using `cl_predict` to predict the class ids or memberships of “new” objects using the partition of the whole feature space induced by the obtained partition of the given objects.

In addition, package `skmeans` provides special methods for the S3 generics `print`, `silhouette` from package `cluster`, and `cl_validity` (providing the “dissimilarity accounted for”) from package `clue` (the latter two take advantage of the special structure of the cosine distance to avoid computing full object-by-object distance matrices, and hence also perform well for large data sets).

3.2. Algorithms

Method "pclus"

This implements the fixed-point algorithm for both the standard and extended spherical k -means problems. For the hard case ($m = 1$), one can optionally attempt further local improvements via Kernighan-Lin chains of first variation single object moves as suggested by Dhillon *et al.* (2002). The chain length employed is specified by control argument `maxchains`.

It is possible to perform several fixed-point runs, either by using control parameter `nruns` to specify the number of runs (which then use the default initialization), or by explicitly giving the starting values via parameter `start`. The default is to perform a single run. The partition returned is the first with the smallest criterion value in the runs performed.

Earlier implementations of this algorithm directly used the general-purpose prototype-based partitioning framework of `pclust` in package `clue`, by providing a `pclust` “family” featuring a dissimilarity function `D` to compute the cosine cross-dissimilarity matrix of objects and prototypes (i.e., the data and prototype matrices) and a consensus function `C` computing a single prototype minimizing $\sum_i \omega_i (1 - \cos(x_i, p))$. However, `pclust` currently always computes optimal prototypes one at a time, and run-time performance can significantly be enhanced by vectorizing these computations (see Section 3.3 for details). Hence, we now use a “stand-alone” implementation, still employing the `skmeans` family for all `skmeans` result objects, in particular as `cl_predict` utilizes its `D` element.

Method "genetic"

This implements a genetic algorithm patterned after the genetic k -means algorithm of [Krishna and Murty \(1999\)](#). Genetic algorithms operate by maintaining a population of solutions of a fixed size and evolving these solutions over a number of generations. In our case, a solution is a membership matrix M (and the corresponding P). In each generation solutions are transformed by mutation and selected for fitness. Typical genetic algorithms also include a cross-over operation that combines two solutions. However, this approach suffers from efficiency problems for clustering, as convergence may take a very long time and crossed-over solutions are unlikely to be better than the starting solutions. As described by [Krishna and Murty \(1999\)](#), instead of the cross-over solution our genetic algorithm applies fixed-point iteration to all mutated solutions before selecting the fittest solution. Fixed-point iteration is computationally cheap and enables the solver to converge much faster than by mutation alone.

In each generation, a mutated copy is created for each solution in the population, doubling the population size. The mutation operation randomly changes cluster membership on a fraction of the objects, which is much simpler and computationally cheaper than the distance-weighted solution presented by [Krishna and Murty \(1999\)](#), but still performs well in our tests. The mutated solutions are then used for fixed-point iteration (the reference uses only a single iteration, we repeat iterations as long as significant improvement can be made). For the selection step, the algorithm computes $F_i = (\Phi(M_i, P_i) - \min_i(\Phi(M_i, P_i))) / (\max_i(\Phi(M_i, P_i)) - \min_i(\Phi(M_i, P_i))) + u_i$, with M_i the memberships of solution i , P_i the corresponding prototypes, and u_i uniformly distributed between 0 and 1, and chooses the top half of solutions ranked by F_i for the next generation. The algorithm terminates if there is no significant improvement over the last generation or the maximum number of generations is reached.

The behavior of the algorithm can be controlled through a number of parameters: `popsize` sets the population size (unless implied by `start`), `reltol` and `maxiter` set termination conditions, and `mutations` gives the probability of mutation in each iteration.

Method "cluto"

This provides an interface to the `vcluster` partitional clustering program from CLUTO, the CLUstering TOolkit by George Karypis ([Karypis 2003](#)). CLUTO is a comprehensive package

for partitional and hierarchical clustering which provides algorithms “optimized for operating on very large datasets both in terms of the number of objects as well as the number of dimensions”, and has been reported to perform very well in many application contexts. It is not open source and can freely be used for educational and research purposes by non-profit institutions and US government agencies only. Standalone programs (`vcluster` and `scluster`) and libraries for Linux (i686/x86_64), Mac OS X (ppc/i386), SunOS, and MS Windows (x86_32, x86_64) can be downloaded via <http://www-users.cs.umn.edu/~karypis/cluto/>. The partitional algorithms of CLUTO use a “randomized incremental optimization algorithm that is greedy in nature, has low computational requirements, and has been shown to produce high-quality clustering solutions” (Karypis 2003). Unfortunately, further algorithmic details are not available due to CLUTO’s closed source nature; from the documentation of the command line arguments `ntrials`, `niter` and `seed` one can infer that several (default: 10) different clustering solutions are computed using a maximum number (default: 10) of refinement iterations, apparently based on suitable random initialization with settable seed (by default, the same seed is used for each invocation). Standard spherical k -means clustering can be performed using `vcluster` (or its library equivalent) with the ‘cos’ similarity and ‘i2’ criterion functions. For convenience reasons, we interface the program rather than the library: using the latter would avoid the cost of file I/O for exchanging data between R and CLUTO, but complicate installation (as the CLUTO libraries and headers need to be available in a place where R can find them when the `skmeans` package is installed).

Method "gmeans"

This provides an interface to the `gmeans` program for partitional clustering developed by Yuqiang Guan as part of his PhD thesis (Guan 2006), which performs k -prototypes partitioning for four different dissimilarity measures d , including cosine dissimilarity. The original source code (released under the GPL with additional citation requirements) is available via <http://userweb.cs.utexas.edu/users/dml/Software/gmeans.html> and known to be compiled using GCC 3.0.3 in Solaris and Linux. However, the code is not ANSI C++ compliant, and hence cannot be compiled using e.g., GCC 4.x. Fortunately, Gmeans was recently compared to CLUTO in the benchmarking study of Tunali, Çamurcu, and Bilgin (2010) who provide modified source code via <http://www.dataminingresearch.com/index.php/2010/06/gmeans-clustering-software-compatible-with-gcc-4/> which can be compiled using current versions of GCC. Gmeans uses the fixed-point algorithm combined with the first variation local improvement strategy of Dhillon *et al.* (2002) (as also available for method “`pclust`”) and provides a choice among six different initialization methods. By default, no first variations are performed, and the initial prototypes are chosen by first determining the spherical 1-means prototype $\sum_i x_i / \|x_i\|$, and then repeatedly picking the x_i most dissimilar to the already chosen prototypes as the next prototype (i.e., by default prototypes are initialized in a deterministic way). There is no library interface, so communication between R and Gmeans requires file I/O.

Method "kmndirs"

This requires package `kmndirs` (available from R-Forge, Maitra, Ramler, and Hornik 2012), which provides an R interface to a suitable modification of the C code for the k -mean-directions algorithm made available as supplementary material to Maitra and Ramler (2010) at <http://pubs.amstat.org/doi/suppl/10.1198/jcgs.2009.08155>. This algorithm fol-

lows the Hartigan-Wong approach to efficiently obtaining k -means partitions by repeatedly considering suitable single object moves (to clusters in the “live set”) in an optimal transfer stage, and potentially updating prototypes and memberships of recently reallocated clusters in a quick transfer stage. The algorithm uses a sophisticated initialization strategy. First, the most widely separated local modes along each data dimension (column of X) are obtained by hierarchical clustering of pruned collections of data quantiles, giving a prototype matrix P_0 . Then, additional prototype matrices P_1, \dots, P_R are obtained by randomly sampling from the rows of X . The P_i for which the criterion value $\min_M \Phi(M, P_i)$ is minimal is selected as starting value for the iteration. In the C code implementing this algorithm, all computations are based on C-level dense matrices.

3.3. Remarks

We already pointed out that in many applications, the data matrix X will be rather large and sparse, so that solvers should be able to employ suitable sparse matrix representations for X (we always use a dense representation for the prototype matrix P). The “native” solvers in package **skmeans** (methods “**pclust**” and “**genetic**”) are capable of dealing with *arbitrary* sparse matrix formats/classes if these provide

- standard methods for subscripting rows, scaling rows via multiplication (from the left) or division (from the right) by a numeric vector, taking (element-wise) squares, and coercion to a dense matrix;
- methods for computing transposed cross-products with a dense matrix (i.e., for computing the matrix product of X and P'), row and column sums.

The former can be made available as methods to the standard generics `[, Ops` (for `*`, `/` and `^`) and `as.matrix`. As `tcrossprod`, `rowSums` and `colSums` are not generic, **skmeans** provides simple (S3) generic wrappers `g_tcrossprod` (with dispatch on the first argument corresponding to the data matrix), `g_row_sums` and `g_col_sums`, with predefined methods for dense matrices, simple triplet matrices from package **slam** (Hornik, Meyer, and Buchta 2011) which are used by the **tm** text mining infrastructure (Feinerer, Hornik, and Meyer 2008) when creating document-term matrices, and the `dgCMatrix` and `dgTMatrix` classes from package **Matrix** (Bates and Maechler 2011). (Packages **slam** and **tm** also provide simple triplet matrix readers for the sparse matrix formats employed by CLUTO and the MC toolkit, Dhillon, Fan, and Guan 2001, respectively.) Support for other formats/classes can be added by providing methods for the above generics.

To use the CLUTO and Gmeans interfaces, an `as.simple_triplet_matrix` coercion method must be available (as we always use the sparse input formats supported by these solvers). For the k -mean-directions interface, `as.matrix` coercion to a dense matrix must be available (as the C implementation internally uses C-level dense matrices).

As we have seen in Section 2, spherical k -means prototypes can be computed via

$$p_j \propto \sum_{i:c(i)=j} \tilde{x}_i, \quad p_j \propto \sum_i \mu_{ij}^m \tilde{x}_i$$

for the hard and soft cases, respectively, i.e., as the column sums of subsets of rows or column sums of scaled rows, respectively. Using the above framework, these can always

be computed via `g_col_sums`, but it may be more efficient to perform these computations on a “lower” level exploiting the internals of sparse matrix representations, and to vectorize computations by simultaneously computing all prototypes. We thus provide a generic `g_col_sums_by_group` for efficiently computing hard prototypes. This has a method for simple triplet matrices based on C code, and a default method individually computing prototypes via `g_col_sums_with_logical_index` (currently non-generic), which in turn provides optimizations for the `dgCMatrix` class. Similarly, soft prototypes can efficiently be computed as the cross-product of the matrices $[\mu_{ij}^m]$ (dense) and the row-normalized data matrix \tilde{X} (possibly sparse). We thus provide a generic `g_crossprod` with dispatch on the second argument, again with methods for dense and simple triplet matrices and the `dgCMatrix` and `dgTMatrix` classes.

3.4. Example

Beginning with “The Wonderful Wizard of Oz”, the Oz books written by L. Frank Baum relating the fictional history of the Land of Oz are among the most popular children’s books ever written. Frank Baum himself authored 14 Oz books. After his death, many additional Oz books were written. Those 26 founded on Baum’s original canon, including 19 books by Ruth Plumly Thompson, and the 14 books by Frank Baum himself give the 40 “official Oz books”.

“The Royal Book of Oz” (Oz book 15) was the first book by Thompson published in 1921, following Baum’s death in 1919. The authorship of this book has long been disputed amongst literature experts. Today, it is commonly attributed to Thompson, as supported by a variety of stylometric analyses.

Package **tm.corpus.Oz.Books**, available from WU Wien’s data package repository at <http://datacube.wu.ac.at/> (and also included in the supplementary materials), provides 21 Oz books in the public domain (all 14 by Baum and 7 by Thompson) as a **tm Corpus** object. The books by Thompson available in this corpus are her first 2 and her last 5. In what follows, we use package **skmeans** for a simple stylometric analysis based on computing spherical k -means partitions of the books in the corpus, illustrating how to obtain a document-term matrix (DTM) suitable for input to **skmeans**, and how to use the result of **skmeans** for subsequent computations.

We start by loading the Oz books data corpus, and using `DocumentTermMatrix` from package **tm** to create a DTM from the corpus.

```
R> data("Oz_Books", package = "tm.corpus.Oz.Books")
R> library("tm")
R> x <- DocumentTermMatrix(Oz_Books,
+   control = list(removePunctuation = TRUE, stopwords = TRUE))
R> x
```

A document-term matrix (21 documents, 39353 terms)

```
Non-/sparse entries: 92643/733770
Sparsity             : 89%
Maximal term length: 2698
Weighting            : term frequency (tf)
```

DTMs in **tm** are based on the simple triplet matrix class from package **slam**, so that using these as inputs to **skmeans** will “automatically” result in efficient representations and computations. Our analysis will be based on binary term weights, i.e., on indicators of the occurrence of terms in documents. Note that the many terms which occur only in a single document (Oz book) are not removed, so that cosine distances between documents will be rather high. As we are also interested in how “sure” Oz book 15 is classified into its cluster, we compute soft spherical k -means partitions with a moderately low softness degree $m = 1.2$ (results obtained are qualitatively similar when varying m around this value). To be able to possibly distinguish the style in the early and late books of Baum and Thompson, respectively, we partition into $k = 4$ classes. We use 20 runs of the fixed-point algorithm to avoid suboptimal solutions due to local optima.

```
R> set.seed(1234)
R> party <- skmeans(weightBin(x), k = 4, m = 1.2, control = list(nruns = 20))
R> party
```

```
A soft spherical k-means partition (degree m = 1.200000) of 21 objects
into 4 classes.
Class sizes of closest hard partition: 7, 2, 7, 5.
Call: skmeans(x = weightBin(x), k = 4, m = 1.2, control = list(nruns = 20))
```

We can then compare the class ids of this partition (precisely, the ids of the closest hard partition, obtained by classifying books into the class with largest membership value) to the “true” authors:

```
R> ids <- abbreviate(unlist(meta(Oz_Books, "Author", type = "local")))
R> table(party$cluster, ids)
```

```
ids
  L.FB RtPT
1     7    0
2     0    2
3     7    0
4     0    5
```

Equivalently, using the high level accessors from package **clue**:

```
R> library("clue")
R> table(cl_class_ids(party), ids)
```

```
ids
  L.FB RtPT
1     7    0
2     0    2
3     7    0
4     0    5
```

Thus, the obtained partition achieves perfect classification. In fact,

```
R> split(cl_class_ids(party), ids)

$L.FB
Oz_Book_01.txt Oz_Book_02.txt Oz_Book_03.txt Oz_Book_04.txt Oz_Book_05.txt
           3           3           3           3           3
Oz_Book_06.txt Oz_Book_07.txt Oz_Book_08.txt Oz_Book_09.txt Oz_Book_10.txt
           3           3           1           1           1
Oz_Book_11.txt Oz_Book_12.txt Oz_Book_13.txt Oz_Book_14.txt
           1           1           1           1

$RtPT
Oz_Book_15.txt Oz_Book_16.txt Oz_Book_29.txt Oz_Book_30.txt Oz_Book_31.txt
           2           2           4           4           4
Oz_Book_32.txt Oz_Book_33.txt
           4           4
```

shows an interesting pattern, with the 2 Thompson clusters cleanly separating her early (first 2) and late (last 5) books, and the 2 Baum clusters similarly separating his books along the time axis.

To assess the “sureness” of the classifications, we inspect the margins (differences between the largest and second largest membership values of the respective books). This gives

```
R> split(round(cl_margin(party), 2), cl_class_ids(party))
```

```
$`1`
[1] 0.41 0.56 0.61 0.49 0.39 0.42 0.55
```

```
$`2`
[1] 0.97 0.97
```

```
$`3`
[1] 0.53 0.02 0.58 0.63 0.70 0.45 0.01
```

```
$`4`
[1] 0.76 0.80 0.77 0.79 0.76
```

Clearly, Oz book number 15 is very strongly classified into its group with Oz book number 16, yet again strengthening the evidence that “The Royal Book of Oz” was indeed written by Thompson, and not by Baum.

Finally, we can use dissimilarity plots ([Hahsler and Hornik 2011](#)) to visualize the quality of the obtained partition (see Figure 1):

```
R> library("seriation")
R> dissplo(t(skmeans_xdist(weightBin(x)), cl_class_ids(party)),
+ options = list(silhouette = TRUE, gp = gpar(cex = 0.7)))
```

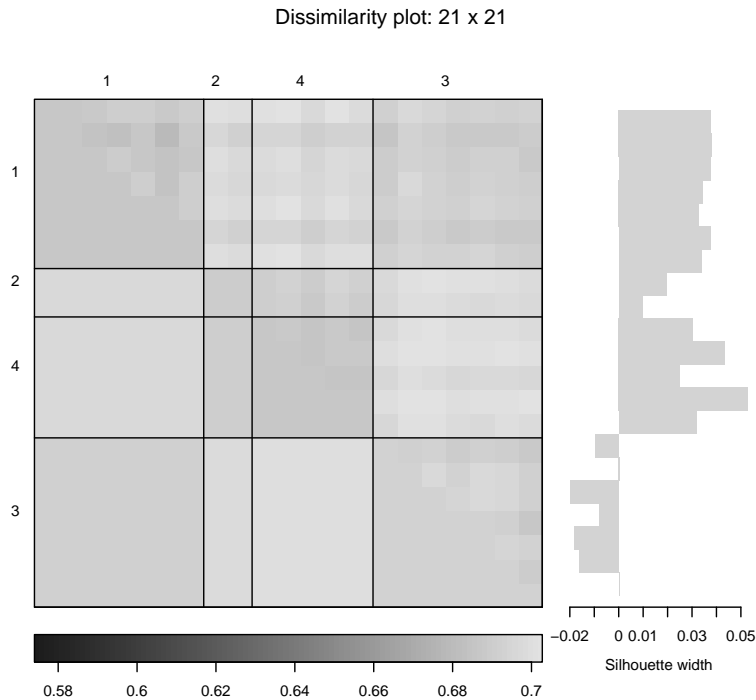


Figure 1: Dissimilarity plot for a soft spherical k -means partition of 21 public domain Oz books into $k = 4$ classes ($m = 1.2$).

This again illustrates that the two Thompson clusters are rather markedly separated from themselves and the Baum clusters, with the last two less well separated from each other.

4. Experiments

4.1. Data

For our experiments we use all 24 datasets which are available from the CLUTO website (<http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto/datasets.tar.gz>). All datasets are document-term matrices storing a bag-of-words representation of the corresponding underlying corpus. Table 1 summarizes their characteristics, listing the number of documents (NR), terms (NC), and number of non-zero entries (NNz), sparsity (in percent) and the memory in kibibyte (KiB, 1024 bytes) that is being used to store the corresponding R object as reported by `object.size()`.

In addition we use the two datasets “20 Newsgroups” and “Waveform” as employed by Tunali *et al.* (2010), and the abstracts from the 2008 Joint Statistical Meetings as used by Maitra and Ramler (2010).

Data	NR	NC	NNz	Sparsity	Size
cacmcisi (SMART)	4663	41681	83181	99.96	1300.72
classic (SMART)	7094	41681	223839	99.92	3498.50
cranmed (SMART)	2431	41681	140658	99.86	2198.79
fbis (TREC, TREC-5)	2463	2000	393386	92.01	6147.66
hitech (TREC, TIPSTER Vol. 3)	2301	126373	346881	99.88	5421.03
k1a (Han <i>et al.</i> 1998)	2340	21839	349792	99.32	5466.51
k1b (Han <i>et al.</i> 1998)	2340	21839	349792	99.32	5466.51
la12 (TREC, TREC-5)	6279	31472	939407	99.52	14679.25
la1 (TREC, TREC-5)	3204	31472	484024	99.52	7563.88
la2 (TREC, TREC-5)	3075	31472	455383	99.53	7116.38
mm	2521	126373	490062	99.85	7658.23
new3 (TREC, TREC-5/6)	9558	83487	2295120	99.71	35862.26
ohscal (Hersh <i>et al.</i> 1994)	11162	11465	674365	99.47	10537.97
re0 (Lewis 1997)	1504	2886	77808	98.21	1216.76
re1 (Lewis 1997)	1657	3758	87328	98.60	1365.51
reviews (TREC, TIPSTER Vol. 3)	4069	126373	781635	99.85	12214.06
sports (TREC, TIPSTER Vol. 3)	8580	126373	1107980	99.90	17313.20
tr11	414	6429	116613	95.62	1823.09
tr12	313	5804	85640	95.29	1339.13
tr23	204	5832	78609	93.39	1229.28
tr31 (TREC, TREC-5/6)	927	10128	248903	97.35	3890.12
tr41 (TREC, TREC-5/6)	878	7454	171509	97.38	2680.84
tr45	690	8261	193605	96.60	3026.09
wap (Han <i>et al.</i> 1998)	1560	8460	220482	98.33	3446.04
ng20 (Mitchell 1999)	18618	25440	318811	99.93	8073.45
jsm2008 (Maitra and Ramler 2010)	2107	3762	84201	98.94	1572.95
waveform (Frank and Asuncion 2010)	5000	21	104828	0.16	1640.18

Table 1: The CLUTO, 20 Newsgroups, 2008 Joint Statistical Meetings, and Waveform data sets and their characteristics: number of rows (NR), number of columns (NC), number of non-zero entries (NNz), sparsity (in percent) and necessary memory to store the corresponding R object (in KiB).

4.2. Setup

We evaluate the methods¹ `cluto`, `genetic`, `gmeans`, and `pclust` both for runtime and cluster performance, by grouping each dataset into $k = 2, \dots, 10$ clusters. In addition we test the runtime impact of the supported matrix formats (dense, simple triplet matrix, and dgTMatrix). Each combination (solver, format, dataset, number of clusters) investigated is run 10 times to reduce the effects of random initializations.

The individual methods are called using default settings subject to following modifications to allow better comparison between the methods and their corresponding criterion values and runtimes.

cluto: Use direct k -way clustering as the method for finding the clusters. Set the seed

¹We do not include `kmndirs` due to memory and stability issues.

of the random number generator to `seed <- as.integer(runif(1) * 2147483647)`. Implemented by setting the `control` argument of the `skmeans` function to `list(control = c("-clmethod=direct", sprintf("-seed=%s", seed)))`.

gmeans: For the initialization method randomly pick vectors as prototypes, and set the number of first variations to 10, via `control = list(control = "-i c -f 10")`.

pclus: Set the number of fixed-point runs to be performed to 12, via `control = list(nruns = 12)`.

In addition for `cluto` and `gmeans` we write out the datasets to disk before actually calling the methods (and only provide pointers to the position via the `ifile` argument) to factor out the overhead induced by their I/O file interfaces.

The experiment is conducted on a quad-core Intel Xeon CPU running at 2.33 GHz equipped with 16 GB RAM.

4.3. Results

To assess (average) clustering performance, we compare the median objective values obtained in the 10 repeated runs for each combination of settings. We take the R-based solvers (`genetic` and `pclus`) using the simple triplet matrix format (results do not change significantly using other formats), and the external solvers (`cluto` and `gmeans`), and for each combination of dataset and number of clusters, rank the solvers according to the criterion value achieved (the lower the rank the better). In the case of ties, corresponding ranks are replaced by their minimum. This gives the cross-tabulation of solvers and ranks as shown in Table 2 with corresponding average ranks

```
cluto genetic gmeans pclus
1.341564 2.267490 3.522634 2.711934
```

This shows that `cluto` finds the best solutions, with the R-based `genetic` and fixed-point solvers in second and third place, respectively. To assess the amount by which `cluto` performs better, we compute the average relative differences of the criterion values obtained by the respective solvers to the ones obtained by CLUTO:

```
genetic gmeans pclus
0.000958808 0.004689899 0.001442029
```

This shows that all solvers perform reasonably well, with the R-based solvers rather close to CLUTO's performance.

	1	2	3	4
<code>cluto</code>	188	34	14	7
<code>genetic</code>	30	122	87	4
<code>gmeans</code>	25	11	19	188
<code>pclus</code>	27	59	114	43

Table 2: Cross-tabulation of solvers and (minimal) ranks of median objective values obtained in 10 repeated runs for each combination of dataset and number of clusters.

To assess (average) clustering performance, we compare the median runtimes values obtained in the 10 repeated runs for each combination of settings. First, we again take the R-based solvers using the simple triplet matrix format, and the external solvers. Results can rather conveniently be summarized by fitting a linear model of the log runtime on the number of clusters and the solver employed, which yields coefficients

(Intercept)	clusters	solvergenetic	solvergmeans	solverpclust
-0.5991801	0.1124755	3.3022608	-0.1410903	2.6198475

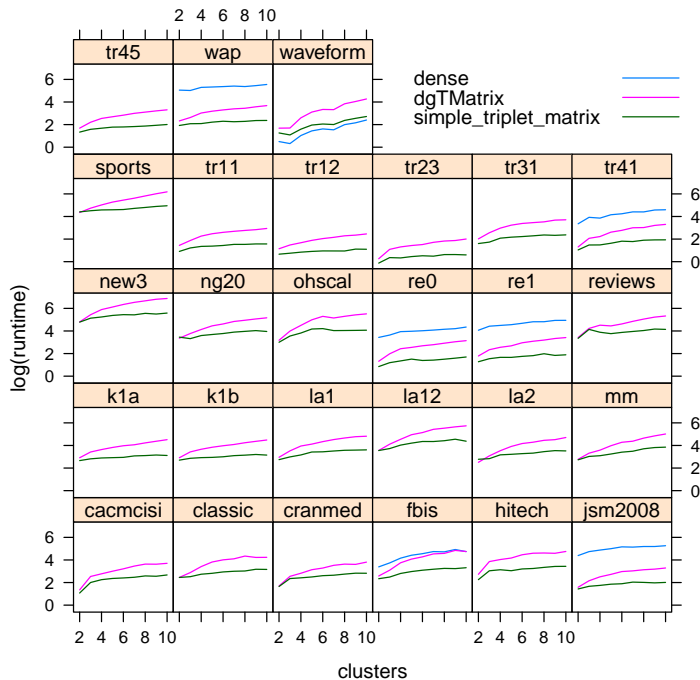
(using the CLUTO log runtime as a baseline) with an R^2 of 66.65%. Thus, on average the `pclust` and `genetic` solvers run about $\exp(2.62) \approx 14$ and $\exp(3.3) \approx 27$ times slower than CLUTO. To compare, direct computation of the average relative difference of the run times obtained by the respective solvers and the ones obtained by CLUTO gives

genetic	gmeans	pclust
28.314724217	-0.009789253	13.732001857

This may seem rather disappointing, in particular given that profiling shows that about 80% of the runtime of the R-based solvers is actually consumed in C code for computing cross-similarities and grouped averages. However, this C code actually spends considerable time validating the arguments passed from R (a somewhat general issue when writing “hybrid” code which moves back and forth between R and C), providing functionality which is reusable rather than optimized for a specific task. Specifically, substantial run-time improvements would be possible by storing the data matrix optimized for row or column access (for more efficiently computing dissimilarities and grouped averages, respectively), or both accesses. We also need to take into account that the amount of computations actually performed may vary significantly between the solvers (as employed). In particular, `gmeans` and `pclust` implement the same fixed-point algorithm, but the results for the former are for a single run with randomly picked vectors as initial prototypes (and with first variation chains of maximal length 10), whereas results for the latter are the best of 12 such runs (with no first variations). Similarly, comparing the run times of the `pclust` and `genetic` solvers and realizing that performing mutation and selection is very cheap, we see that with the chosen defaults (population sizes and mutation rates), the latter performs about twice as many fixed-point iterations as the former. See Section 3.2 for comments on the lack of available algorithmic details for the `cluto` solver: the documentation would seem to imply that each run performs at most 100 refinement iterations, where we found that e.g., `pclust` (as employed) performed substantially more fixed-point iterations.

The preceding discussion explains why `gmeans` runs the fastest, but gives the worst performance. It might also be taken to suggest attempting to modify the defaults of the R-based solvers so that substantially fewer iterations are performed to obtain results of “similar” quality (criterion value). We think differently: after all, the real task is to optimize the criterion function, so in principle we should use available resources at our discretion to find the best approximate solutions we can obtain within the resource constraints.

For the R-based solvers, we can also investigate the impact of using different storage formats for the data matrix. Figure 2 shows the median log runtimes using `pclust`. Not surprisingly, using dense matrices performs worst (but quite interestingly, best for the waveform data set, which is “very dense”). Due to memory limitations, results for dense matrices are only available

Figure 2: Median log run-times for method `pclus`.

for the “small” datasets. Using simple triplet matrices clearly performs best. We observe a rather non-linear growth pattern for the `dgTMatrix` format. Results using the `genetic` solver, or `dgCMatrix` instead of `dgTMatrix` are equivalent. Due to this pattern, fitting linear models of log runtime on the number of clusters and the solver and formats employed yields rather low R^2 values (in the 30 % range). Instead, directly computing the average relative differences of the run times obtained by using the `dgTMatrix` and dense formats and the ones using the simple triplet matrix format gives

```
dgTMatrix    dense
 1.589124 12.343601
```

which under-estimates the speedups observed for larger numbers of clusters: using e.g., only the results for $k > 6$ gives the following average relative differences:

```
dgTMatrix    dense
 2.290238 13.157902
```

Finally, we investigate the extent to which the best solutions obtained can further be improved by suitable chains of first variation moves. We use maximal chain lengths of 1 (`v_lih`), 10 (`v_lihc_10`), and 25 (`v_lihc_25`) (alternatively, one might consider lengths corresponding to fractions of the number of points to be clustered). Average relative improvements achieved for the respective criterion values are shown in Table 3 with the fractions of results for which improvements occurred listed in Table 4. Remember that `gmeans` was run with the number of first variations set to 10: nevertheless, single element move improvements were possible

Solver	v_lih	v_lihc_10	v_lihc_25
<code>cluto</code>	3.883046E-06	7.779577E-06	1.121617E-05
<code>genetic</code>	1.090766E-04	1.169718E-04	1.284170E-04
<code>gmeans</code>	3.714942E-06	2.427019E-05	4.285163E-05
<code>pclus</code>	5.206787E-04	5.741406E-04	5.914045E-04

Table 3: Average relative improvements achieved for the respective criterion values.

Solver	v_lih	v_lihc_10	v_lihc_25
<code>cluto</code>	0.4074074	0.5231481	0.5370370
<code>genetic</code>	0.8935185	0.9027778	0.9027778
<code>gmeans</code>	0.5138889	0.6064815	0.6342593
<code>pclus</code>	0.9444444	0.9444444	0.9444444

Table 4: Fractions of results for which improvements occurred.

for about half of the results. For `cluto`, results are comparable to those for `gmeans`; hence, one might suspect that `cluto` also attempts local improvements. The results of the R-based solvers can “typically” be improved (note that whereas the fractions may remain the same, the amount of improvement still grows with the lengths of the chains employed). The success rates seem to indicate that one should always try to improve obtained solutions using chains with lengths not below 10.

5. Conclusion

Spherical k -means clustering is a central technique for addressing current data analysis challenges, especially in the context of large collections of text documents. Solving spherical k -means clustering problems corresponds to finding optimal group memberships employing the cosine dissimilarity measure.

We presented a computational environment with R-based implementations of a fixed-point and a genetic algorithm, and interfaces to two well-known external solvers (`CLUTO` and `Gmeans`). The infrastructure is highly extensible, with support for arbitrary sparse matrix formats, and allows the customization of initialization and local improvement strategies.

A large scale benchmark experiment analyzing the performance and efficiency of the available solvers showed that all four presented approaches scale well and can be used for realistic data sets with acceptable clustering performance. The external solvers `Gmeans` and `CLUTO` are both very fast, with `CLUTO` typically providing better solutions. The genetic algorithm finds excellent solutions but has the longest runtime, whereas the fixed-point algorithm is a very good all-round approach. In general we recommend to use local improvements to further optimize computed cluster assignments, which in our benchmark experiment worked best with chains of length at least 10.

References

Banerjee A, Dhillon IS, Ghosh J, Sra S (2005). “Clustering on the Unit Hypersphere Using von

- Mises-Fisher Distributions.” *Journal of Machine Learning Research*, **6**, 1345–1382. URL <http://jmlr.csail.mit.edu/papers/v6/banerjee05a.html>.
- Bates D, Maechler M (2011). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.0-1, URL <http://CRAN.R-project.org/package=Matrix>.
- Belacel N, Hansen P, Mladenovic N (2002). “Fuzzy J -Means: A New Heuristic for Fuzzy Clustering.” *Pattern Recognition*, **35**, 2193–2200.
- Bezdek JC (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York.
- Blei DM, Lafferty JD (2009). “Topic Models.” In A Srivastava, M Sahami (eds.), *Text Mining: Classification, Clustering, and Applications*, pp. 71–94. Taylor and Francis.
- D’haeseleer P (2005). “How Does Gene Expression Clustering Work?” *Nature Biotechnology*, **23**, 1499–1501.
- Dhillon IS, Fan J, Guan Y (2001). “Efficient Clustering of Very Large Document Collections.” In R Grossman, C Kamath, V Kumar, R Namburu (eds.), *Data Mining for Scientific and Engineering Applications*, pp. 357–381. Kluwer Academic Publishers.
- Dhillon IS, Guan Y, Kogan J (2002). “Iterative Clustering of High Dimensional Text Data Augmented by Local Search.” In *Proceedings of the Second IEEE International Conference on Data Mining*, pp. 131–138. Maebishi, Japan. URL http://www.cs.utexas.edu/users/inderjit/public_papers/iterative_icdm02.pdf.
- Dhillon IS, Modha DS (2001). “Concept Decompositions for Large Sparse Text Data Using Clustering.” *Machine Learning*, **42**(1), 143–175.
- Feinerer I, Hornik K, Meyer D (2008). “Text Mining Infrastructure in R.” *Journal of Statistical Software*, **25**(5), 1–54. URL <http://www.jstatsoft.org/v25/i05/>.
- Frank A, Asuncion A (2010). “UCI Machine Learning Repository.” URL <http://archive.ics.uci.edu/ml>.
- Griffiths TL, Steyvers M, Tenenbaum JB (2007). “Topics in Semantic Representation.” *Psychological Review*, **114**(2), 211–244.
- Guan Y (2006). *Large-Scale Clustering: Algorithms and Applications*. Ph.D. thesis, The University of Texas, Austin.
- Hahsler M, Hornik K (2011). “Dissimilarity Plots: A Visual Exploration Tool for Partitional Clustering.” *Journal of Computational and Graphical Statistics*, **10**(2), 335–354.
- Han EH, Boley D, Gini M, Gross R, Hastings K, Karypis G, Kumar V, Mobasher B, Moore J (1998). “WebACE: A Web Agent for Document Categorization and Exploration.” In *AGENTS ’98: Proceedings of the second international conference on Autonomous agents*, pp. 408–415. ACM, New York. doi:10.1145/280765.280872.

- Hersh W, Buckley C, Leone TJ, Hickam D (1994). “OHSUMED: An Interactive Retrieval Evaluation and New Large Test Collection for Research.” In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 192–201. Springer-Verlag, New York.
- Hornik K (2005). “A CLUE for CLUster Ensembles.” *Journal of Statistical Software*, **14**(12), 1–25. URL <http://www.jstatsoft.org/v14/i12/>.
- Hornik K, Meyer D, Buchta C (2011). *slam: Sparse Lightweight Arrays and Matrices*. R package version 0.1-22, URL <http://CRAN.R-project.org/package=slam>.
- Karatzoglou A, Feinerer I (2010). “Kernel-Based Machine Learning for Fast Text Mining in R.” *Computational Statistics & Data Analysis*, **54**(2), 290–297.
- Karypis G (2003). “CLUTO: A Clustering Toolkit.” *Technical Report 02-017*, Department of Computer Science, University of Minnesota. URL <http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto/manual.pdf>.
- Krishna K, Murty MN (1999). “Genetic K -Means Algorithm.” *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, **29**(3), 433–439.
- Leisch F (2006). “A Toolbox for K -Centroids Cluster Analysis.” *Computational Statistics & Data Analysis*, **51**(2), 526–544.
- Lewis D (1997). “Reuters-21578 Text Categorization Test Collection.” URL <http://www.daviddlewis.com/resources/testcollections/reuters21578/>.
- Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002). “Text Classification Using String Kernels.” *Journal of Machine Learning Research*, **2**, 419–444. doi:<http://jmlr.csail.mit.edu/papers/v2/lodhi02a.html>.
- Maitra R, Ramler IP (2010). “A K -Mean-Directions Algorithm for Fast Clustering of Data on the Sphere.” *Journal of Computational and Graphical Statistics*, **19**(2), 377–396.
- Maitra R, Ramler IP, Hornik K (2012). *kmndirs: k-Mean-Directions Algorithm*. R package version 0.0-0/r628, URL <http://R-Forge.R-project.org/projects/kmndirs/>.
- Manning CD, Raghavan P, Schütze H (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge. URL <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>.
- Mitchell T (1999). “20 Newsgroups.” UCI KDD Archive. URL <http://kdd.ics.uci.edu/databases/20newsgroups/>.
- R Development Core Team (2012). *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/>.
- Salton G, Wong A, Yang CS (1975). “A Vector Space Model for Automatic Indexing.” *Communications of the ACM*, **18**(11), 613–620.
- SMART (1999). “SMART Information Retrieval System.” URL <ftp://ftp.cs.cornell.edu/pub/smart/>.

Tan PN, Kumar V, Srivastava J (2004). “Selecting the Right Objective Measure for Association Analysis.” *Information Systems*, **29**(4), 293–313.

TREC (2010). “Text REtrieval Conference.” URL <http://trec.nist.gov/>.

Tunali V, Çamurcu AY, Bilgin TT (2010). “An Empirical Comparison of Fast and Efficient Tools for Mining Textual Data.” Presented at “1st International Symposium on Computing in Science and Engineering, held by Gediz University in Kusadasi, Turkey. URL <http://www.dataminingresearch.com/index.php/2010/07/cluto-vs-gmeans-empirical-comparison/>.

Zhao Y, Karypis G (2004). “Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering.” *Machine Learning*, **55**, 311–331.

Affiliation:

Christian Buchta, Kurt Hornik, Martin Kober
Institute for Statistics and Mathematics
WU Wirtschaftsuniversität Wien
Augasse 2–6
1090 Wien, Austria
E-mail: Christian.Buchta@wu.ac.at,
Kurt.Hornik@R-project.org,
Martin.Kober@wu.ac.at

Ingo Feinerer
Institute of Information Systems
Technische Universität Wien
1040 Wien, Austria
E-mail: Ingo.Feinerer@tuwien.ac.at