Ckmeans.1d.dp: Optimal *k*-means Clustering in One Dimension by Dynamic Programming

by Haizhou Wang and Mingzhou Song

Abstract
The heuristic *k*-means algorithm, widely used for cluster analysis, does not guarantee optimality. We developed a dynamic programming algorithm for optimal one-dimensional clustering. The algorithm is implemented as an R package called **Ckmeans.1d.dp**. We demonstrate its advantage in optimality and runtime over the standard iterative *k*-means algorithm.

Introduction

Cluster analysis offers a useful way to organize and represent complex data sets. It is used routinely for data analysis in fields such as bioinformatics. The *k*-means problem is to partition data into *k* groups such that the sum of squared Euclidean distances to each group mean is minimized. However, the problem is NP-hard in a general Euclidean space, even when the number of clusters *k* is 2 (Aloise et al., 2009; Dasgupta and Freund, 2009), or when the dimensionality is 2 (Mahajan et al., 2009). The standard iterative *k*-means algorithm (Lloyd, 1982) is a widely used heuristic solution. The algorithm iteratively calculates the within-cluster sum of squared distances, modifies group membership of each point to reduce the within-cluster sum of squared distances, and computes new cluster centers until local convergence is achieved. The time complexity of this standard *k*-means algorithm is $O(qknp)$, where q is the number of iterations, k is the number of clusters, n is the sample size, and p is the dimensionality (Manning et al., 2008).

However, the disadvantages of various heuristic *k*-means algorithms in repeatability, optimality, and runtime may limit their usage in medical and scientific applications. For example, when patients are clustered according to diagnostic measurements on expression of marker genes, it can be critical that a patient consistently falls into a same diagnostic group to receive appropriate treatment, regardless of how many times the clustering is applied. In this case, both cluster repeatability and optimality are important. The result of heuristic *k*-means clustering, heavily dependent on the initial cluster centers, is neither always optimal nor repeatable. Often one restarts the procedure a number of times to mitigate the problem. However, when k is big, the number of restarts for *k*-means to approach an optimal solution can be prohibitively high and may lead to a substantial increase in runtime. As clustering is often a preprocessing step in data analysis or modeling, such inadequacy can pose a nuisance for further steps.

In response to this challenge, our objective is to develop a practical and convenient clustering algorithm in R to guarantee optimality in a one-dimensional (1-D) space. The motivation originated from discrete system modeling such as Boolean networks (Kauffman, 1969, 1993) and generalized logical networks (Song et al., 2009), where continuous values must be converted to discrete ones before modeling. Our algorithm follows a comparable dynamic programming strategy used in a 1-D quantization problem to preserve probability distributions (Song et al., 2010), the segmented least squares problem and the knapsack problem (Kleinberg and Tardos, 2006). The objective function in the *k*-means problem is the sum of squares of within-cluster distances. We present an exact dynamic programming solution with a runtime of $O(n^2k)$ to the 1-D *k*-means problem.

We implemented the algorithm in the R package **Ckmeans.1d.dp** (Song and Wang, 2011) and evaluated its performance by simulation studies. We demonstrate how the standard *k*-means algorithm may return unstable clustering results, while our method guarantees optimality. Our method is implemented in C++ to take advantage of the practical speedup of a compiled program. This C++ implementation is further wrapped in an R package so that it can be conveniently called in R.

Optimal 1-D clustering by dynamic programming

We first define the *k*-means problem. Let $x_1, \ldots, x_n$ be an input array of n numbers sorted in non-descending order. The problem of 1-D *k*-means clustering is defined as assigning elements of the input 1-D array into k clusters so that the sum of squares of within-cluster distances from each element to its corresponding cluster mean is minimized. We refer to this sum as within-cluster sum of squares, or *withinss* for short.

We introduce a dynamic programming algorithm to guarantee optimality of clustering in 1-D. We define a sub-problem as finding the minimum *withinss* of clustering $x_1, \ldots, x_i$ into m clusters. We record the corresponding minimum *withinss* in entry $D[i, m]$ of an $n+1$ by $k+1$ matrix D. Thus $D[n, k]$ is the minimum *withinss* value to the original problem. Let j be the index of the smallest number in cluster m in an optimal solution to $D[i,m]$. It is evident that $D[j-1,m-1]$ must be the optimal *withinss* for the first $j-1$ points in $m-1$ clusters, for otherwise one would have a better solution to $D[i,m]$. This establishes the optimal substructure for dynamic programming and leads to the recurrence equation

$$D[i,m] = \min_{m \le j \le i} \left\{ D[j-1,m-1] + d(x_j,\ldots,x_i) \right\}, \qquad 1 \le i \le n, 1 \le m \le k$$

where $d(x_j,\ldots,x_i)$ is the sum of squared distances from $x_j,\ldots,x_i$ to their mean. The matrix is initialized as $D[i,m] = 0$, when $m = 0$ or $i = 0$.

Using the above recurrence, we can obtain $D[n,k]$, the minimum *withinss* achievable by a clustering of the given data. In the meanwhile, $D[n,m]$ gives the minimum *withinss* if all n numbers are clustered into m groups. By definition each entry requires $O(n^2)$ time to compute using the given recurrence if $d(x_j,\ldots,x_i)$ is computed in linear time, resulting in $O(n^3k)$ total runtime. However, $d(x_j,\ldots,x_i)$ can be computed in constant time in the recurrence. This is possible because $d(x_j,\ldots,x_i)$ can be computed progressively based on $d(x_{j+1},...,x_i)$ in constant time. Using a general index from 1 to i, we iteratively compute

$$d(x_1,\ldots,x_i) = d(x_1,\ldots,x_{i-1}) + \frac{i-1}{i}(x_i - \mu_{i-1})^2$$
$$\mu_i = \frac{x_i + (i-1)\mu_{i-1}}{i}$$

where μ_i is the mean of the first i elements. This iterative computation of $d(x_j,\ldots,x_i)$ reduces the overall runtime of the dynamic programming algorithm to $O(n^2k)$.

To find a clustering of data with the minimum *withinss* of $D[n,k]$, we define an auxiliary n by k matrix B to record the index of the smallest number in cluster m:

$$B[i,m] = \operatorname*{argmin}_{m \le j \le i} \left\{ D[j-1,m-1] + d(x_j,\ldots,x_i) \right\}, \qquad 1 \le i \le n, 1 \le m \le k$$

Then we backtrack from $B[n,k]$ to obtain the starting and ending indices for all clusters and generate an optimal solution to the k-means problem. The backtrack is done in $O(k)$ time.

The space complexity of the dynamic programming is $O(nk)$ because we use an $(n+1) \times (k+1)$ matrix D to record minimum *withinss* and an $n \times k$ matrix B for backtracking.

Implementation

We implemented this dynamic programming algorithm and created an R package **Ckmeans.1d.dp**. To take advantage of the runtime speed up by a compiled language over R, an interpreted language, we developed the dynamic programming solution in C++. We compiled the C++ code to a binary dynamic library, which can be called within R through function `Ckmeans.1d.dp()` provided in the package.

Performance evaluation and comparison with the standard k-means

Simulated data sets

We simulated data sets containing clusters using 1-D Gaussian mixture models. In a Gaussian mixture model, the number of components is the true number of clusters. The mean of each Gaussian component is randomly sampled from the uniform distribution from -1 to 1 and the standard deviation of a component is uniformly distributed from 0 to 0.2.

Optimality

We first compared the optimality of `Ckmeans.1d.dp()` and the standard k-means method on 1-D data. We used the Hartigan and Wong (1979) implementation of k-means, as provided by the `kmeans()` function in R. The core of the `kmeans()` function is also implemented in C/C++. As `Ckmeans.1d.dp()` guarantees optimality of clustering, we define the relative difference from the `kmeans()` clustering result to the optimal value produced by `Ckmeans.1d.dp()` as

$$\frac{\textit{withinss}(\texttt{k-means}) - \textit{withinss}(\texttt{Ckmeans.1d.dp})}{\textit{withinss}(\texttt{Ckmeans.1d.dp})}$$

which measures deviation of the k-means result from the optimal value.

The data sets were randomly generated from Gaussian mixture models with 2 to 50 components. We ran `Ckmeans.1d.dp()` and `kmeans()` on input data given the correct number of clusters k. The relative difference in *withinss* from the `kmeans()` to the optimal value produced by `Ckmeans.1d.dp()` is shown as a function of k in Figure 1.

Although one hopes to get better clusters by restarting k-means several times, the simulation suggests that it still cannot guarantee optimality when there are many clusters in the data. This clearly demonstrates the advantage of `Ckmeans.1d.dp()` in optimality.

Runtime

We then compared the empirical runtime of `Ckmeans.1d.dp()` with the standard k-means method on 1-D data. Using 1-D Gaussian mixture models, we generated five input arrays of size 100, 1,000, 10,000, 100,000, and 1,000,000, respectively. The Gaussian mixture models had 2 to 50 components.

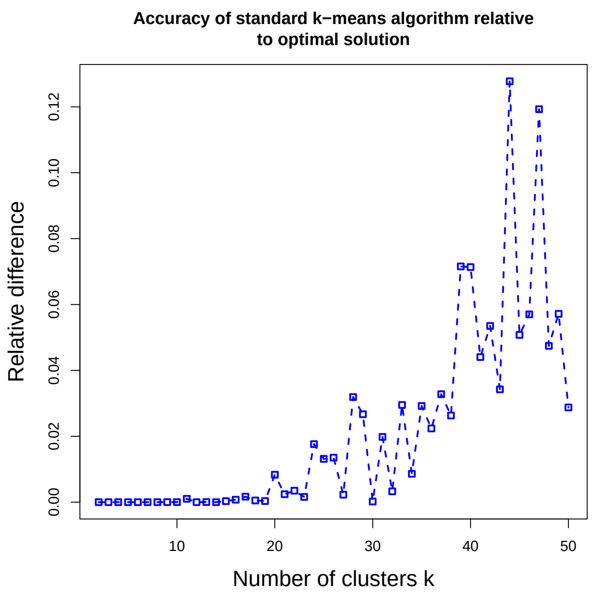

Figure 1: The accuracy of `kmeans()` becomes worse as the number of clusters increases. Accuracy is indicated by the relative difference in *withinss* from `kmeans()` to the optimal value returned by `Ckmeans.1d.dp()`. The input data sets of size 10,000 were sampled from Gaussian mixture models of 2 to 50 components. `kmeans()` was set to restart 20 times for each data set.

We ran both programs on each input array ten times to obtain average empirical runtimes in three different settings. All simulations were run on a desktop computer with an Intel Core i7 860 processor and 8GB memory, running OpenSUSE 11.3 and R 2.11.1.

In the first setting (Figure 2), runtime is obtained as a function of input data size from 100 to 1,000,000 for both `Ckmeans.1d.dp()` and `kmeans()` without constraints on optimality. The number of components in the Gaussian mixture models is fixed to 2.

According to Figure 2, as the input size increases, the runtime of `Ckmeans.1d.dp()` increases faster than the runtime of `kmeans()`. However, the result returned by `kmeans()` may not be optimal (the restart of `kmeans()` was set to 1). Without any constraints on optimality, `kmeans()` demonstrates an advantage in runtime over `Ckmeans.1d.dp()` of runtime quadratic in n.

In the second setting (Figure 3), runtime is obtained as a function of number of clusters for `Ckmeans.1d.dp()` and `kmeans()` without constraints on optimality. All input data sets are of the same size 10,000 and were generated from Gaussian mixture models with 2 to 25 components.

In Figure 3, the runtime of `Ckmeans.1d.dp()` increases linearly with the number of clusters k, as does the runtime of `kmeans()`. `kmeans()`, without any constraints on optimality, still shows an advantage in absolute runtime over `Ckmeans.1d.dp()`.

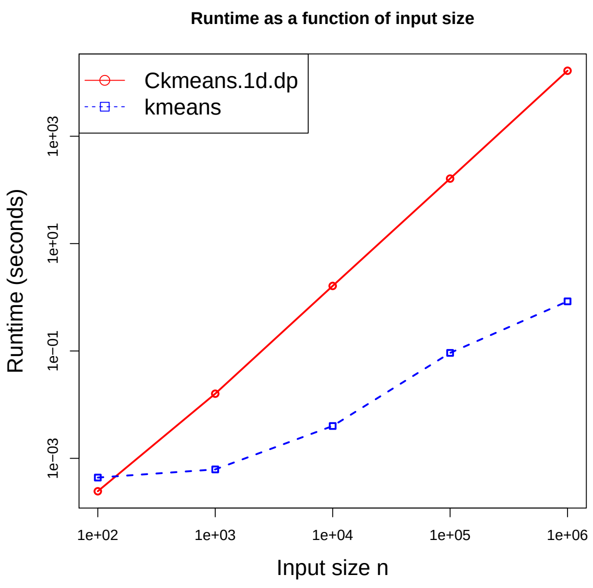

Figure 2: Runtime as a function of input size for `Ckmeans.1d.dp()` and `kmeans()`. `kmeans()` shows an advantage in runtime without any constraints on optimality. The number of clusters is 2 and there is no restart for k-means.

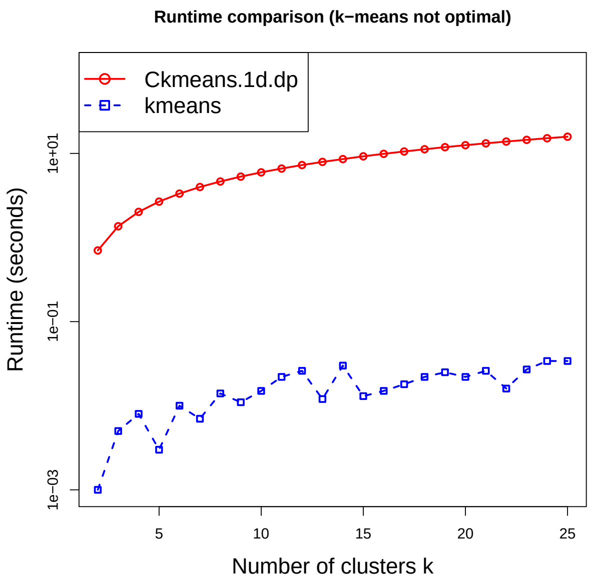

Figure 3: Comparison of runtime as a function of number of clusters between `Ckmeans.1d.dp()` and `kmeans()`. The input data are mixed Gaussian of fixed size 10,000 but with an increasing number of components, representing the clusters in data. Although the k-means algorithm took less runtime, it was run with no restart and thus may not be optimal.

In the third setting (Figure 4), we plot the runtime of `Ckmeans.1d.dp()` and `kmeans()` as a function of the number of clusters, obtained from exactly the same input data in the second setting, but with a constraint on the optimality of k-means such that its *withinss* has a relative difference less than 10^{-6} from the optimal value.

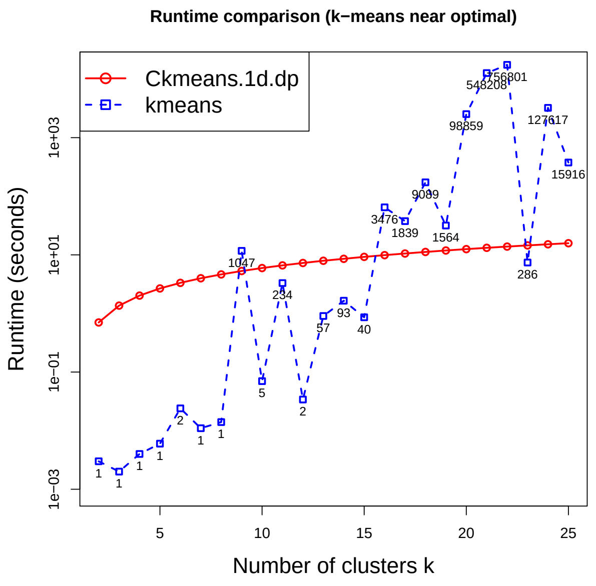

Figure 4: Runtime as a function of number of clusters with restart of `kmeans()` to guarantee a relative difference less than 10^{-6} to the optimal value. The sample size is always 10,000. The number next to each blue square is the number of restarts needed for `kmeans()`. The runtime of `kmeans()` appears to grow exponentially and that of `Ckmeans.1d.dp()` always grows linearly with the number of clusters.

Since `Ckmeans.1d.dp()` returns an optimal result in only one run, its runtime is the same as setting 2. On the other hand, `kmeans()` was restarted a number of times in order to approach an optimal solution within the given tolerance. The number of restarts increases substantially to produce a nearly optimal solution as the number of clusters k increases. As shown in Figure 4, the runtime of `Ckmeans.1d.dp()` increases linearly with k, but the runtime of `kmeans()` appears to increase exponentially with k. The runtime of `kmeans()` almost always far exceeds `Ckmeans.1d.dp()` when k is above 15. This suggests an advantage of `Ckmeans.1d.dp()` over `kmeans()` in both runtime and optimality when k is big.

The R source code for the simulation studies is available from `http://www.cs.nmsu.edu/~joemsong/R-Journal/Ckmeans-Simulation.zip`.

Introduction to the R package Ckmeans.1d.dp

The **Ckmeans.1d.dp** package implements the dynamic programming algorithm for 1-D clustering that we described above. In the package, the `Ckmeans.1d.dp()` function performs the clustering on a 1-D input vector using a given number of clusters.

The following example illustrates how to use the package. Figure 5 visualizes the input data and the cluster result obtained by `Ckmeans.1d.dp()` in this example.

```
# a one-dimensional example
# with a two-component Gaussian mixture model
x <- rnorm(50, mean = 1, sd = 0.3)
x <- append(x, rnorm(50, sd = 0.3) )
result <- Ckmeans.1d.dp(x, 2)
plot(x, col = result$cluster)
abline(h = result$centers, col = 1:2, pch = 8,
       cex = 2)
```

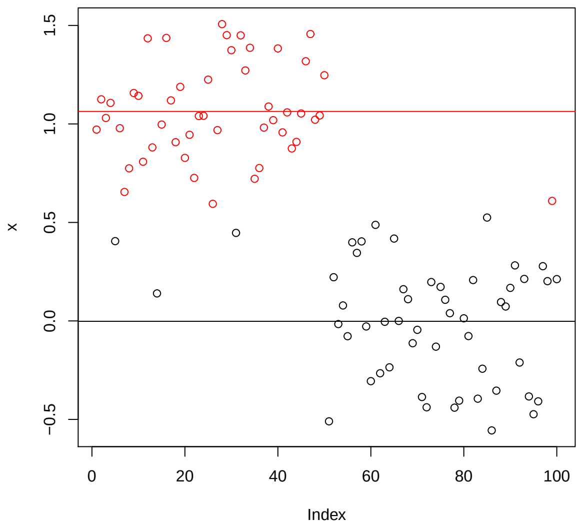

Figure 5: Two clusters obtained from applying function `Ckmeans.1d.dp()` on input x sampled from a Gaussian mixture model with two components. The horizontal axis is the index number of each point in x. The two clusters are indicated using red and black circles. The two horizontal lines represent the centers of each cluster.

Summary

The main purpose of our work is to develop an exact solution to 1-D clustering in a practical amount of time, as an alternative to heuristic k-means algorithms. We thus developed the **Ckmeans.1d.dp** package using dynamic programming to guarantee clustering optimality in $O(n^2k)$ time. The algorithm is implemented in C++ with an interface to the R language. It is provided as a package to R and has been tested under recent operating system versions of Windows, Linux and MacOS. By simulation studies, we demonstrated its advantage over the standard k-means algorithm when optimality is required.

There are two limitations of our **Ckmeans.1d.dp** method. It only applies to 1-D data and its quadratic runtime in the sample size may not be acceptable in all applications.

However, where applicable, the impact of our method is its optimality and repeatability. As our simulation studies show, when the number of clusters is big, our method not only guarantees optimality but also has a fast runtime. In this situation, our method is preferred. We applied our **Ckmeans.1d.dp** to quantize biological data in the DREAM Challenges (Prill et al., 2010) for qualitative dynamic modeling using generalized logical networks (Song et al., 2009).

It is of great interest if the 1-D dynamic programming strategy can be extended to multiple dimensional spaces so that clustering can be done in polynomial time to k. However, this seems to us to remain open indefinitely.

Acknowledgement

We gratefully acknowledge the financial support to the reported work by the grant number HRD-0420407 from the U.S. National Science Foundation.

Bibliography

D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, 2009.

S. Dasgupta and Y. Freund. Random projection trees for vector quantization. *IEEE T. Inform. Theory*, 55 (7):3229–3242, 2009.

J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *J. Roy. Stat. Soc. C-App.*, 28(1):100–108, 1979.

S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theoretical Biology*, 22(3):437–467, 1969.

S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, June 1993.

J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, USA, 2006.

S. P. Lloyd. Least squares quantization in PCM. *IEEE T. Inform. Theory*, 28:129 – 137, 1982.

M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. In *WALCOM '09: Proceedings of the 3rd International Workshop on Algorithms and Computation*, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.

C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

R. J. Prill, D. Marbach, J. Saez-Rodriguez, P. K. Sorger, L. G. Alexopoulos, X. Xue, N. D. Clarke, G. Altan-Bonnet, and G. Stolovitzky. Towards a rigorous assessment of systems biology models: The DREAM3 challenges. *PLoS ONE*, 5(2):e9202, 02 2010.

M. Song and H. Wang. *Ckmeans.1d.dp: Optimal distance-based clustering for one-dimensional data*, 2011. URL `http://CRAN.R-project.org/package=Ckmeans.1d.dp`. R package version 2.2.

M. Song, C. K. Lewis, E. R. Lance, E. J. Chesler, R. K. Yordanova, M. A. Langston, K. H. Lodowski, and S. E. Bergeson. Reconstructing generalized logical networks of transcriptional regulation in mouse brain from temporal gene expression data. *EURASIP J. Bioinform. Syst. Biol.*, 2009:1–13, 2009.

M. Song, R. M. Haralick, and S. Boissinot. Efficient and exact maximum likelihood quantisation of genomic features using dynamic programming. *Int. J. Data Min. Bioinform.*, 4(2):123–141, 2010.

Mingzhou Song
Department of Computer Science
New Mexico State University
United States
`joemsong@cs.nmsu.edu`

Haizhou Wang
Department of Computer Science
New Mexico State University
United States
`hwang@cs.nmsu.edu`