Clustering Application Benchmark

Oğuz Altun[1], Nilgün Dursunoğlu[2], M.Fatih Amasyalı[3]

[1,3]Yıldız Technical University, Computer Engineering Department, Turkey

[2]Garanti Teknoloji, Turkey

E-mail: [1,3]{oguz, mfatih}@ce.yildiz.edu.tr, [2]nilgunerd@garanti.com.tr

Abstract

An application benchmark based on a set of clustering algorithms is described in this paper. The code provided complies with ANSI C specifications, as a result is highly portable. The benchmark has been tested on various platforms using different compilers.

1. Introduction

The application benchmark presented in this paper is based on a set of clustering algorithms. A *cluster* is therefore a collection of objects which are “similar” to each other and are “dissimilar” to the objects that belong to the other clusters. The clustering algorithms’ aim is to find clusters from unlabeled data. Clustering algorithms can be applied in many fields [1]. In marketing applications, the customer groups with similar behavior are found given by clustering a large database of customer data that contain their properties and past buying records. In biology, plants and animals are clustered according to their features. City planners identify groups of houses according to their house type, value and geographical location. The earthquake scientist clusters observed earthquake epicenters to identify dangerous zones. In Web domain, the document classification and clustering web log data to discover groups of similar access patterns are very popular applications.

2. Description of the Benchmark

1. Usage of the Code

The code is written in ANSI C programming language. Our benchmark can be integrated to other C/C++ programs by calling cluster_benchmark() function in the code, or by compiling supplied clusbenc.c program into an executable and running it. To call cluster_benchmark() you must include "clusbenc.h" header file in your file. The first parameter is the path to the input file, and the second parameter is the number of iterations. If you want your benchmark runs take longer increase this parameter.

The supplied clusbenc.c program also serves as an example of the usage of cluster_benchmark().

A simple makefile for gcc compiler is also provided with the code.

2. Clustering Algorithms

The benchmark code runs 6 clustering algorithms sequentially (K-means online, K-means batch, SOM-1D, SOM-2D, Hierarchical K-means online and Hierarchical SOM-1D). The algorithms as used in the benchmark are defined below [2]:

(i) *K-Means online:*

1- Initialize K group centroids.

2- For each sample

  a- Assign a sample to the group that has the closest centroid.

  b- Recalculate the new position of the assigned centroid.

3- Repeat Step 2 until the centroids no longer move or the maximum iteration number has been reached.

(ii) *K-Means batch:*

1- Initialize K group centroids.

2- Assign each sample to the group that has the closest centroid.

3- When all samples have been assigned, recalculate the new positions of the K centroids.

4- Repeat Steps 2 and 3 until the centroids no longer move or the maximum iteration number has been reached.

(iii) *SOM 1-dimension:*

1- Initialize K group centroids and connections between centroids in 1 dimensional space.

2- For each sample

  a- Assign a sample to the group that has the closest centroid.

  b- Recalculate the new positions of the assigned centroid and its neighbors in 1 dimensional space according to their distances from the sample.

3- Repeat Step 2 until the centroids no longer move or the maximum iteration number has been reached.

(iv) *SOM 2-dimension:*

The only difference between SOM 1-dimension and 2-dimension is the connections of centroids. In SOM 2-dimensional, the centroids have neighbors in 2-dimensions.

(v) *Hierarchical K-Means online:*

The algorithm divides the samples recursively into clusters. The K-means online algorithm is used by setting k to two in order to divide the samples into two clusters. Then, the two clusters are divided again into two clusters by setting k to two. The recursion terminates when the cluster number is equal to k [3].

(vi) ***Hierarchical SOM 1-dimension:***

The only difference between Hierarchical K-means Online and Hierarchical SOM 1-dimension is the clustering algorithms which divides the samples into clusters.

3. **The .dst File format**

The .dst file format is a very simple text file format. First the height (number of rows) and then the width (number of columns) of the array should be written. The rest is the data from left to right, top to bottom. The text file must conform to these rules:

- First value in the file must be an unsigned integer equal to the height (number of rows) of the 2 dimensional array.
- Second value must be an unsigned integer equal to the width (number of columns on each row) of the 2 dimensional array.
- The cell values must be real (double) values. No comment is allowed in the file.
- Values must be separated by white space. No characters, except numbers and white space, are allowed.

4. **Construction of the Input Data Sets**

The benchmark code can accept any dataset in .dst format, users can easily create and use their own data set. The clusbenc.c program we suplied looks for a data set named "data.dst" in the working directory. This can be easily changed by giving a different data set path argumant to the cluster_benchmark() function.

In our experiments we applied benchmark code to a document dataset. The dataset is constructed from 920 Turkish news texts belonging to 4 classes (economy, sport, politic, popular). A document/newstext is represented in a word-based space with 11954 dimensions. The resulting dataset “data.dst” has 920 columns and 11954 rows where each row shows the passing count of a word in the corresponding document.

5. **The Output of the Benchmark**

Considering the portability issues, to get the elapsed CPU time used by the benchmark the `clock` function is preferred. This function is specified by the ANSI C standard. The `clock` function is called at the beginning and end of the interval to be timed. Subtracting these values and diving by `CLOCKS_PER_SEC` (the number of clock ticks per second), the elapsed CPU time is calculated.

After executing the clustering algorithms, the code prints out total execution time spent in seconds.

6. Portability Issues

To ensure portability, only ANSI C functions are used in the code. All file names are in the 8.3 naming convention in case operating system has such restriction.

The bundled code includes the full documentation of the benchmark code in .html format, prepared with Doxygen documentation system [4].

7. Tested Platforms and Results

We have tested the code on various platforms for both data sets that are submitted with the code. Table 1 gives the benchmark results on various platforms using different compilers for the data set "data.dst".

Table 1. The Results of the Clustering Benchmark for Various Platforms

Architecture	**Operating System**	**Compiler**	**Total Time (In Seconds)**
Intel Pentium 4, CPU 3.00GHz, 504 MB RAM	MS. WIN. XP PRO, version 2002, SP2	MS. Visual Studio 2005 Professional Edition, Debug mode	16.391
"	"	MS. Visual Studio 2005 Professional Edition, Release Mode	11.578
"	"	MS. Visual C++ 6.0, release mode	17,906
"	"	MS. Visual C++ 6.0, release mode	6.062
"	"	Borland C++ Builder Enterprise Suite, Version 6	16.844
"	"	Gcc 3.4.2, thread model win32	15.969
"	"	Gcc 3.4.2, thread model win32, with best optimization	15.845
Intel Xeon MP CPU 3.66 GHz, 3,95 GB RAM	Suse Linux 9.3	Gcc 3.3.5, thread model posix	15.720
Intel Pentium 4 CPU 1.70 GHz, 440 MB RAM	Debian Linux Sid	Gcc 4.4.4, thread model posix	36,260

1. **References**

[1] http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/

[2] E. Alpaydın, Machine Learning, MIT Press, 2004

[3] http://gecco.org.chemie.uni-frankfurt.de/hkmeans/H-k-means.pdf

[4] Dimitri van Heesch, Doxygen documentation system, http://www.stack.nl/~dimitri/doxygen