Sampling Clustering

Ching Tarn*, Yinan Zhang and Ye Feng
School of Computer Science and Technology, Shandong University, China
+i@ctarn.io

Abstract—We propose an efficient graph-based divisive cluster analysis approach called sampling clustering. It constructs a lite informative dendrogram by recursively dividing a graph into subgraphs. In each recursive call, a graph is sampled first with a set of vertices being removed to disconnect latent clusters, then condensed by adding edges to the remaining vertices to avoid graph fragmentation caused by vertex removals. We also present some sampling and condensing methods and discuss the effectiveness in this paper. Our implementations run in linear time and achieve outstanding performance on various types of datasets. Experimental results show that they outperform state-of-the-art clustering algorithms with significantly less computing resources requirements.

Index Terms—clustering, graph-based, hierarchical, divisive

I. INTRODUCTION

Cluster analysis is a basic common unsupervised learning approach and widely used in various fields such as data mining and computer vision. It categorizes a set of objects into groups (or clusters) based on the similarities between them to help understand and analyze data. Clusters are aggregations of objects that intra-cluster similarities are higher than inter-cluster similarities.

Due to the long history of research on clustering, a large number of algorithms have been proposed. There exist several common limitations.

A. Classical Partitional Methods

Classical center-based algorithms like k-Means [1] use central vectors to partition the data space and may lack the ability to find arbitrary shaped clusters. They can also be very sensitive to the center initialization. Distribution-based algorithms including Gaussian Mixture Models have similar shortcomings. In addition, with the increase of the dimension, the number of parameters to be determined also increases. The running time and memory requirements may become unacceptable.

Density-based clustering methods define clusters as contiguous regions with high density. DBSCAN [2] assumes that clusters have homogeneous densities. A global threshold needs to be pre-selected to distinguish between high density and low density areas. It simply judges whether two regions are connected from the local density of area between them, which can result in “single-link effect” and lead to an undesirable connectivity. [3] Another critical problem is that there may not be a global threshold to separate all clusters in practical applications. It makes such tasks unsolvable to DBSCAN.

Algorithms above generate disjoint partitions of the dataset, and nesting is not allowed. However, many data in reality are hierarchical, and it is hard, or impossible, to find a single reasonable and unambiguous division of the data. In such cases, a possible way is to generate multiple partitions on the same dataset with different granularities to reconstruct the hierarchy. It requires multiple runs and is difficult to handle.

B. Agglomerative and Divisive Methods

Hierarchical algorithms are traditionally divided into two categories, agglomerative and divisive methods. Such algorithms generate hierarchical structures called dendrograms, which are much more informative than non-nested partitions. A dendrogram can also be converted into partitions with different number of clusters as needed without multiple runs.

Typically, agglomerative methods initially treat each single object as a cluster, or start with a large number of tiny clusters, and merge them in pairs. The dendrogram can be very deep and large, which makes it difficult to analyze. In fact, low level branches are not so meaningful, and truncation is often used to make the dendrogram easier to analyze, which is also a challenging task.

In addition, because clusters are merged in pairs, dendrograms are usually binary and thus cannot represent the actual structures accurately. For example, assuming a cluster consisting of more than two identical subclusters, the cluster can’t be built by merging all subclusters simultaneously and thus the dendrogram largely depends on the order of selections.

To select a pair of clusters for the next merge, the similarities between every two clusters must be calculated. The complexity of an agglomerative method is often very high, usually $O(N^3)$, and can be unacceptable for a mass of data.

In general, a divisive method initially groups all objects into a cluster, and divides each one into two subclusters recursively. The stopping condition (e.g., the commonly used minimum cluster size) is usually hard to be defined properly. It results in similar problems with agglomerative methods.

C. Graph-based Methods

Many methods require the data to be spatial, which limits their applications. Graph-based methods use similarity graphs, and usually transform spatial data into nearest neighbor graphs (e.g., k-NN, mutual k-NN [4] and XNN [5]). Although the transformation may cause loss of information, processing on graphs still brings many significant benefits. 1) There are concepts in graph theory, like reachability, that can measure the similarities between objects better than traditional measures (e.g., the Euclidean distance). 2) Querying the most similar objects in a graph is often faster than others. It is very useful
for iterative algorithms, like [6], that the data are modified in each iteration and new nearest neighbors need to be queried multiple times. 3) It has been shown in [7] that nearest neighbor graphs are proper and efficient representations for data lying on a low dimensional manifold embedded in a high dimensional space.

With the considerations and inspired by [6], we propose a graph-based divisive clustering approach called sampling clustering. It recursively divides a graph into subgraphs through removing vertices from the graph. The approach has the following advantages.

- It generates a non-binary dendrogram, which is much more representative than non-nested partitions and binary dendrograms.
- The division stops in time, and undesirable splits are less. The dendrogram is small and easy to analyze.
- It works well and outperforms state-of-the-art algorithms on various types of datasets.
- Only a few hyper-parameters are required and each one doesn’t play a key role.
- It requires significantly less computing resources, and usually runs in linear time. It is also easily parallelizable.
- It is very simple, and can be extended to handle domain-specific data.

The paper is organized as follows. We present the details of the approach in Section II. Section III shows the experimental results. The conclusions are in Section IV.

II. THE APPROACH

Given a graph of the data, the approach constructs a dendrogram through dividing a graph into subgraphs recursively. In each recursive call, the graph is sampled first to disconnect latent subclusters. The selected vertices are removed from the graph along with edges. After each sampling, the density of the graph may decrease and connections between vertices become weaker. Multiple samplings may divide the graph into a lot of tiny parts unnecessarily. To avoid fragmentation of a cluster, the condensing is applied on the sampled graph. This operation is to find new nearest neighbors of each remaining vertex and connect them with their new neighbors respectively to fill vacancies around them.

The recursive process terminates if no vertex or all vertices of a graph are removed by sampling.

After sampling, each vertex to be deleted from the graph is associated with a remaining nearest vertex if the process is not terminated. Otherwise, the call returns the set of all vertices of the graph along with vertices associated, perhaps indirectly, with them. Each vertex can always find a nearest one because the graph is connected and there exists at least one remaining vertex. We divide the input into subgraphs without sampling if it is not connected.

An overview is as Algorithm 1.

Algorithm 1 Sampling Clustering

```
procedure CLUSTER(G = (V, E))
    R ← SAMPLE(G) ⊿ the set of vertices to be removed
    if R = ∅ or R = V then
        T ← the vertices associated with V
        return V ∪ T ⊿ a leaf of the dendrogram
    else
        for all v ∈ R do
            u ← the nearest vertex ∈ V\R to v
            associate v to u
        end for
        G ← CONDENSE(G, R) ⊿ the new graph
        D ← ∅ ⊿ an empty node
        for all subgraph S of G do
            d ← CLUSTER(S)
            D ← D ∪ {d} ⊿ append d to D as a child
        end for
        return D
    end if
end procedure
```

In the rest of this section, we present a set of sampling and condensing methods, and introduce some evaluation measures. We also provide a fine-tuning method and simple algorithms...
to convert a dendrogram into a partition. In the end we briefly
discuss the complexity of the approach.

For the convenience of the following description, assuming
a directed graph $G = (V, E)$, we first list definitions we use.

**Definition 1** (neighbor). $u$ is a neighbor of $v$ if and only if
$v, u \in E$.

**Definition 2** (neighborhood). The neighborhood of $v$ is the
set of all neighbors of $v$, denoted as $N_v$.

In addition, we denote $N_v \cup \{v\}$ as $N_v^+$.  

**Definition 3** (dendrogram). A dendrogram is a tree that all
leaves are exclusive subsets of $V$.

### A. Sampling

Sampling vertices from a graph leads to several benefits.
1) By reducing the amount of vertices, both analyzing and
computing become easier. With a proper subset of the graph,
its main structural information can be well maintained. 2) Multiple samplings can break the weak connections caused
by noise between components. It avoids common problems,
like “single-link effect”, of linkage-methods and makes the
approach more tolerant to noise. 3) More importantly, the
sequence of separations contains rich structural information
and can be used to reconstruct the hierarchical structure of
the data.

The sampling algorithm is described as Algorithm 2. It uses
a measure $s(\cdot)$ to calculate a score of each vertex that indicates
whether it should be deleted or not, and return the set of vertices with the lowest score, that is, to be removed. We
take sampling rate $r$ to control the number of vertices to be removed.

**Algorithm 2** Sampling

```plaintext
for all vertex $v \in V$ do
    $v$.score $\leftarrow s(v)$
end for
return the $r|V|$ vertices with the lowest score
```

A basic sampling measure $s_r$ is assigning a random score
to each vertex. Another type of methods tends to remove
vertices at the junction of clusters instead of vertices in core
areas, so the latent clusters can be disconnected quickly and
undesirable divisions can be avoided. Such methods can be
boundary detection based, like [8]. It is shown in [8] that
a vertex at the junction usually has a smaller indegree than
others. In particular, for a $k$-nn graph, the out-degree of each
vertex is $k$, and the indegree of a vertex at the boundary is
usually less than $k$, as shown in Fig. 2.

**Definition 4** (indegree sampling),

$$s_i(v) = |\{u|v \in N_u\}|$$

Another similar one, defined as Definition 5, identifies
boundaries based on the number of mutual neighbors of each
vertex.

**Definition 5** (mutual neighbor sampling),

$$s_m(v) = |\{u|v \in N_u \land u \in N_v\}|$$

To measure the effectiveness of a sampling method, first we
formally define the vertices at the junction as Definition 6.

**Definition 6** (positive vertex). A vertex $v$ is at the junction if
and only if $\exists u \in N_v$, that $v$ and $u$ belong to different clusters.
A vertex not at the junction is called a positive vertex.

A sampled graph should be more separable. We use the
proportion of positive vertices of the graph to measure its
separability.

**Definition 7** (vertex positivity). The proportion of positive
vertices of a graph is called its vertex positivity, denoted as
$P_v$.

To avoid the effect of the size of a neighborhood deceasing,
we determine whether a vertex is positive before sampling and
only recalculate the proportion after that. Obviously, sampling
randomly doesn’t change the proportion of positive vertices.

### B. Condensing

A sampled graph is usually sparser, and connections be-
tween vertices turn to be weaker. Multiple samplings may
break the graph into pieces. We condense the graph by adding
edges to it to avoid graph fragmentation caused by vertex
removals. For a sampled nearest neighbor graph, we search
new $t$ nearest vertices of each vertex and connect them with
their new neighbors respectively.

For a vertex $v$, we use BFS, breadth-first search, starting
from $v$ to find a set of candidates $C_v$. To avoid searching too
depth unnecessarily, we limit the depth to $d$. It stops if $|C_v| \geq t$
and the searching depth is greater than $d$. A measure $c(\cdot)$ is
used to calculate a score of each candidate which indicates the
distance, or dissimilarity, between it and $v$.

The candidates with high similarity become its new neigh-
bors. A new graph is built using edges weighted by scores.
The algorithm is formally described as Algorithm 3.
Algorithm 3 Condensing

\[ E' \leftarrow \emptyset \quad \triangleright \text{ an empty edges set} \]

for all vertex \( v \in V \setminus R \) do

\[ C \leftarrow \emptyset \]

initialize a BFS iterator starting from \( v \)

while \(|C| < t \) or the depth of BFS \( \leq d \) do

\[ u \leftarrow \text{the next vertex to be visited} \]

if \( u = NULL \) then break \( \triangleright \) there is no reachable vertex.

else if \( u \in V \setminus R \) then

\[ u\cdot\text{score} \leftarrow c(v,u) \]

\[ C \leftarrow C \cup \{u\} \]

end if

end while

\[ S \leftarrow \text{the max}(\{|C|, t\}) \text{ vertices in } C \text{ with the lowest score} \]

for all vertex \( u \in S \) do

\[ E' \leftarrow E' \cup \{< u,v,u\cdot\text{score}> \} \quad \triangleright \text{ weighted edges} \]

end for

end for

return \( G = (V \setminus R, E') \)

The simplest measure \( c_v \) is based on the BFS visiting sequence. We simply assign the index to a candidate as its score. In this case, BFS starts from the lightest edge so that the condensing can be more stable.

Another measure is shared neighbor, denoted as \( c_s \). It has been shown in [11] that a high similarity between two neighborhoods also indicates that the two vertices are similar. We use Jaccard index, also known as Intersection over Union (IoU), to measure the similarity between two neighborhoods.

Definition 8 (Jaccard index). \( \text{IoU}(a,b) = \frac{|N^+_a \cap N^+_b|}{|N^+_a \cup N^+_b|} \).

For spatial data, distance metrics (e.g., Euclidean distance \( c_d \)) can also be used in some applications.

Effective condensing methods should maintain the structure of the graph well, and this can be considered from two aspects.

1) Inter-cluster connections should be avoided. A vertex should belong to the same cluster with its neighbors to make clusters separable. 2) To avoid a cluster to be divided into pieces, vertices in it need to be strongly connected.

Fig. 4 shows that graph-based measures are usually more effective than geometry-based.

Definition 9 (positive edge). An edge \( <u,v> \) is positive if and only if \( u \) and \( v \) belong to the same cluster.

Definition 10 (edge positivity). The proportion of positive edges of a graph is called its edge positivity, denoted as \( P_e \).

For the second one, the average connectivity of graph [12], defined as follows, is used to measure the strength of a cluster’s internal connections. It can also be used to evaluate the stability of a sampling method.

Definition 11 (graph connectivity). The connectivity of a graph is measured with

\[ C = \frac{\sum_{u,v \in V} f(u,v)}{\binom{|V|}{2}}, \]

where \( f(u,v) \) is the value of maximum flow from \( u \) to \( v \).

Fig. 4 shows that graph-based measures are usually more effective than geometry-based.
connections are extremely weak. The intra-cluster connections are destroyed and the clusters are divided into pieces.

Fig. 5 shows that, the connectivity of graph decreases rapidly after about 6 iterations if $c_v$ is employed. In fact, if a $k$-nn graph is not sampled after each iteration, tiny parts are generated even faster, while $c_v$ doesn’t change the graph at all if $t = k$.

C. Truncation

Truncation is used to reduce the number of leaves of a dendrogram and generate a partition from it. We provide two simple truncation algorithms.

The first algorithm strictly conforms to the structure of the dendrogram. A node can be merged if and only if it is not a leaf and its all children are leaves. Only merging all children of one node together simultaneously is allowed.

The size of a node is defined as the total number of objects in its descendants, or itself. We first merge the smallest node that all children are leaves. The algorithm stops if the next merge causes the number of leaves to be less than the desired value $n$, or the largest $n$ leaves have contained more than $\alpha |V|$ objects. It is described as Algorithm 4.

The second algorithm allows two leaves belonging to the same parent to be merged together directly. A leaf can be moved down if it have a brother and the brother is not a leaf. The detailed description is shown in Algorithm 5. It generates a more balanced partition with the number of clusters being precisely controlled. However, it may also lead to structural information loss and doesn’t work well on unbalanced data.

D. Fine-Tuning

We also introduce a simple method called smoothing to adjust the dendrogram. It is based on the observation that, firstly, some associating trees, which are generated after sampling, may cross together slightly, which results in the fact that a small number of vertices, especially at the junction, and their neighbors are grouped into different clusters; secondly, a few isolated vertices may become clusters unexpectedly, and such tiny leaves should also be removed.

We simply relabel each vertex with the most-common cluster in its neighborhoods. This operation can be repeated multiple times, and our experiments show that it improves the quality on most datasets and converges fast.

E. Complexity Analysis

Finally, we briefly analyze the complexity of the approach. Given a $k$-nn graph $G = (V,E)$, first we discuss the complexity of each step in one recursive call.

1) Sampling: Assuming a sampling measure $s(\cdot)$ with a time complexity of $f_s$, calculating the scores costs $O(|V|f_s)$. Selection algorithms are used to calculate the subset of vertices with the lowest score, and can be finished at the cost of $O(|V|)$, so the complexity of sampling is $O(|V|f_s)$.

Lemma 1. The complexity of sampling is $O(|V|f_s)$.
Lemma 2. The complexity of associating is $O(|V|)$.

3) Condensing: Because we limit the searching depth to $d$, for each vertex, the number of candidates does not exceed $\max(t, \sum_{i=1}^{d} k^i)$, and thus calculating and sorting the scores cost only $O(f_c)$, where $f_c$ is the complexity of the measure.

Lemma 3. The complexity of condensing is $O(|V| f_c)$.

Computing strongly or weakly connected components also runs in linear time, so the complexity of a recursive call is $O(|V|(f_s + f_e))$.

Because the total size of all subgraphs is $r |V|$ and $0 < r < 1$, the complexity of our approach is $O(|V|(f_s + f_e))$.

Theorem 1. The complexity of sampling clustering is $O(|V|(f_s + f_e))$.

The complexity of any measure introduced above is $O(1)$, and thus the complexity of any implementation based on them is linear.

III. EXPERIMENTS

A. Comparison with other algorithms

1) Datasets: We follow the experiments in [13], Robust Continuous Clustering, and use the datasets [14]–[19] preprocessed and publicly provided by the authors. A brief summary is shown in Table I. More details can be found in [13].

2) Baselines: We compare Sampling Clustering (SC) with both partitional methods, including $k$-Means++ (KM) [20], Mean Shift (MS) [21], Gaussian Mixture Models (GMM), Affinity Propagation (AP) [22] and Robust Continuous Clustering (RCC, RCC-DR), and hierarchical methods, including three classical agglomerative algorithms (AC-Average, AC-Complete, AC-Ward), Graph Degree Linkage (GDL-U, AGDL) [23] and Hierarchical DBSCAN (HD) [24].

For RCCs (RCC, RCC-DR) and GDLs (GDL-U, AGDL), we use implementations publicly provided by the authors, and for others, we use the scikit-learn and scikit-learn-contrib.

3) Measures: We use AMI$_{max}$ [25] in the scikit-learn to evaluate all algorithms. The results evaluated with NMI are also provided in the appendix.

4) Settings:

a) Sampling Clustering: The Euclidean distance metric is used to build a 16-neighbor graph of each dataset. We use indegree sampling $s_i$ and the condensing method based on BFS visiting sequence $c_v$. Although $c_v$ is not stable and may divide a graph into pieces after multiple runs, it still be usable and very effective, and thus we run condensing on the input using $c_v$ only once to improve the quality of the graph. The parameters are fixed as $r = 0.2$, $t = 16$. We do BFS on the graph with the sampled vertices being removed and limit the searching depth to 2. Graphs are divided into strongly connected components. Smoothing runs 16 times before and after truncation. We use hard truncation on unbalanced or noise-rich datasets (YTF, Reuters, RCV1, Shuttle and Mice Protein) and soft truncation on others. For hard truncation, $\alpha$ is fixed as 0.8.

b) KM, GMM: Run each algorithm 4 times.

c) MS: quantile $\in \{0.001, 0.003, 0.01, 0.03, 0.1\}$.

d) AP: max_iter $= 1000$, convergence_iter $= 100$, damping $= 0.9$.

e) RCCs: max_iter $= 100$, inner_iter $= 4$. The weighted graphs are provided by the authors.

f) GDLs: $a \in \{10^{-2}, 10^{-1.5}, \cdots, 10^2\}$, $K = 20$.

g) HD: min$_{cluster}$.size $\in \{2^1, 2^2, \cdots, 2^{\lfloor \log_2\bar{n} \rfloor}\}$, where $\bar{n}$ is the average size of ground-truth clusters.

The default setting is used if not mentioned. For algorithms that run multiple times, including KM, GMM, MS, GDLs and HD, we use the best results.

5) Results: We use a computer with an Intel Core i7-6770HQ CPU (2.60GHz × 8) and 31.3 GiB memory. The results are shown in Table II. Some algorithms may require too much memory on a dataset and thus are not applicable, marked as $MLE$.

We also compare the computing resources costs of each algorithm. Considering the scalability of some algorithms, as shown in Table III, we only evaluate them on the RCV1.

It shows that the implementation based on Sampling Clustering achieves the best results on three datasets and requires significantly less computing resources, while each one of the other algorithms, including state-of-the-art algorithms like RCCs and GDLs, works best on at most one dataset.

GMM failed on high dimensional datasets. The MNIST is the biggest dataset, and all agglomerative methods and affinity propagation are not applicable on it. GDLs and AP also failed on another big dataset, the Shuttle.

B. Comparison of sampling and condensing measures

We evaluate the measures on four datasets, including MNIST, Pendigits, YaleB and YTF. The results are shown in Table IV. The $\bar{n}$ indicates the average size of ground-truth clusters of each dataset.

It shows that $c_v$ usually leads to bad results, expected that $s_r$ is used, because sampling randomly slows down the process.

TABLE I DATASETS

| Dataset   | Instances | Dimensions | Classes |
|-----------|-----------|------------|---------|
| MNIST     | 70,000    | 784        | 10      |
| COIL100   | 7,200     | 49,152     | 100     |
| YaleB     | 2,414     | 32,256     | 38      |
| Reuters   | 10,056    | 9,075      | 40      |
| RCV1      | 9,082     | 2,000      | 50      |
| Pendigits | 10,000    | 2,000      | 4       |
| Shuttle   | 58,000    | 9,075      | 40      |
| Mice Protein | 1,077  | 77         | 8       |

2 The measure used in [13] is AMI$_{max}$. 
of generating tiny components, while other methods tend to remove vertices at the junction and vertices in the core areas are divided into pieces very fast.

With the increase of data volume, the differences between them become more and more dramatic. On the MNIST, that each cluster contains 7,000 digits, $s_r$, $c_r$, and $c_d$ are poorly effective, while all measures are almost equally effective on the YaleB and YTF.

C. Robustness

We analyze the robustness to the sampling rate $r$, condensing size $t$, and the $k$-nn graph size $k$, and vary them in ranges $r \in \{0.1, 0.2, \ldots, 0.9\}$ and $t, k \in \{4, 6, \ldots, 30\}$ respectively. Other settings are the same as III-A4a.

Due to the limited space and the similarity between $s_i$ and $s_m$, we only consider six settings, including ($s_r$, $c_r$), ($s_r$, $c_d$), ($s_i$, $c_r$), ($s_i$, $c_d$), ($s_i$, $c_r$) and ($s_i$, $c_d$), and evaluate them on the Pendigits. The results are shown in Fig. 6, 7 and 8 respectively. Obviously, each parameter does not play a critical role in the approach for most measures in a wide range.

D. Effectiveness of smoothing

We also test the effectiveness of smoothing. Instead of running 16 times before and after truncation as the previous experiments, to make the effect on partitions more obvious, we only run once before truncation to remove tiny clusters and 16 times after that. The results (Fig. 9) shows that it is effective on most datasets and converges very fast.

E. Visualization

The visualizations of clustering results on the MNIST is shown in Fig. 10 and 11. The number on a branch indicates the size of it, and the images are 9 random samples in it. We also visualize the results in III-A using t-SNE [26], as shown in the appendix.
As can be seen, without any truncation, the dendrogram is quite small, compared with other binary outputs of classical hierarchical methods. The truncated dendrogram shows that it first separates 1s from the whole, and 0s then. The remaining digits are roughly divided into three groups, \{8\}, \{3, \{5, 6\}\}, and \{2, \{4, \{7, 9\}\}\}.

**IV. CONCLUSIONS**

We present an efficient graph-based divisive clustering approach which is shown to be effective on various types of data. We also take an introductory discussion on strategies to divide faster and to avoid undesirable divisions. We hope this work can inspire further cluster analysis research.

**REFERENCES**

[1] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[2] M. Ester, H. Kriegel, J. Sander, and X. Xiaowei, “A density-based algorithm for discovering clusters in large spatial databases with noise,” pp. 226–231, 1996.

[3] H. Kriegel, P. Kroger, J. Sander, and A. Zimek, “Density-based clustering,” *Wiley Interdisciplinary Reviews-Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[4] M. Brito, E. Chvez, A. Quiroz, and J. Yukich, “Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection,” *Statistics & Probability Letters*, vol. 35, no. 1, pp. 33 – 42, 1997.

[5] P. Franti, R. Mariescu-Istodor, and C. Zhong, “Xnn graph,” in *Structural, Syntactic, and Statistical Pattern Recognition*, A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, and R. Wilson, Eds. Cham: Springer International Publishing, 2016, pp. 207–217.

[6] N. Bar, H. Averbuch-Elor, and D. Cohen-Or, “Border-peeling clustering,” *CoRR*, vol. abs/1612.04869, 2016.

[7] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural Computation*, vol. 15, no. 6,
Fig. 11. Visualization of the dendrogram on the MNIST


**APPENDIX**

| Dataset   | KM | MS | GMM | AP  | RCC | RCC-DR | AC-A | AC-C | AC-W | GDL-U | AGDL | HD | SC |
|-----------|----|----|-----|-----|-----|--------|------|------|------|-------|------|----|----|
| MNIST     | 0.500 | 0.430 | 0.311 | MLE | 0.893 | 0.830 | MLE | MLE | MLE | MLE | MLE | 0.356 | 0.895 |
| COIL100   | 0.840 | 0.852 | MLE | 0.843 | 0.963 | 0.963 | 0.687 | 0.754 | 0.862 | 0.961 | 0.961 | 0.905 | 0.894 |
| YTF       | 0.789 | 0.831 | MLE | 0.783 | 0.892 | 0.890 | 0.583 | 0.681 | 0.808 | 0.699 | 0.692 | 0.846 | 0.881 |
| YaleB     | 0.658 | 0.742 | MLE | 0.799 | 0.978 | 0.975 | 0.231 | 0.479 | 0.774 | 0.967 | 0.967 | 0.696 | 0.914 |
| Reuters   | 0.535 | 0.000 | 0.538 | 0.503 | 0.556 | 0.553 | 0.533 | 0.392 | 0.487 | 0.514 | 0.515 | 0.413 | 0.538 |
| RCV1      | 0.511 | 0.000 | 0.566 | 0.355 | 0.138 | 0.437 | 0.142 | 0.108 | 0.375 | 0.116 | 0.182 | 0.301 | 0.169 |
| Pendigits | 0.682 | 0.738 | 0.715 | 0.648 | 0.845 | 0.851 | 0.659 | 0.584 | 0.728 | 0.593 | 0.593 | 0.738 | 0.860 |
| Shuttle   | 0.216 | 0.375 | 0.356 | MLE | 0.488 | 0.546 | 0.040 | 0.039 | 0.246 | MEL | MEL | 0.622 | 0.530 |
| Mice Protein | 0.479 | 0.600 | 0.426 | 0.593 | 0.662 | 0.636 | 0.377 | 0.324 | 0.514 | 0.513 | 0.513 | 0.602 | 0.581 |

MLE: Memory Limit Exceeded.

**Fig. 12.** Visualization of Results on Datasets.