Efficient Parameter-free Clustering Using First Neighbor Relations

M. Saquib Sarfraz[1,2], Vivek Sharma[1], Rainer Stiefelhagen[1]
[1]Karlsruhe Institute of Technology
[2]Daimler TSS, Germany
{firstname.lastname}@kit.edu

Abstract

We present a new clustering method in the form of a single clustering equation that is able to directly discover groupings in the data. The main proposition is that the first neighbor of each sample is all one needs to discover large chains and finding the groups in the data. In contrast to most existing clustering algorithms our method does not require any hyper-parameters, distance thresholds and/or the need to specify the number of clusters. The proposed algorithm belongs to the family of hierarchical agglomerative methods. The technique has a very low computational overhead, is easily scalable and applicable to large practical problems. Evaluation on well known datasets from different domains ranging between 1077 and 8.1 million samples shows substantial performance gains when compared to the existing clustering techniques. [code release]

1. Introduction

Discovering natural groupings in data is needed in virtually all areas of sciences. Despite half a decade of research in devising clustering techniques there is still no universal solution that 1.) can automatically determine the true (or close to true) clusters with high accuracy/purity; 2.) does not need hyper-parameters or a priori knowledge of data; 3.) generalizes across different data domains; and 4.) can scale easily to very large (millions of samples) data without requiring prohibitive computational resources. Clustering inherently builds on the notion of similarity. All clustering techniques strive to capture this notion of similarity by devising a criterion that may result in a defined local optimal solution for groupings in the data. Well known center-based methods (e.g., Kmeans) iteratively assign points to a chosen number of clusters based on their direct distance to the cluster center. Agglomerative clustering methods merge points based on predefined distance thresholds. More recent methods build similarity graphs (e.g., spectral clustering techniques) from the pairwise distances of the points and solve a graph partitioning problem by using these distances as edge weights and the sample points as nodes. All existing clustering techniques use some form of prior knowledge/assumptions on defining the similarity goal to obtain specific groupings in the data. This prior knowledge comes in the form of setting number of clusters in advance or setting distance thresholds or other hyper-parameters that render a user defined notion of similarity for obtaining groupings. These choices are subjective and must change when the underlying data distribution changes. This means that these parameters are not stable and need to be adjusted for each data set. This makes the clustering problem very hard as no standard solution can be used for different problems.

In this paper, we describe an efficient and fully parameter-free unsupervised clustering algorithm that does not suffer from any of the aforementioned problems. By "*fully*", we mean that the algorithm does not require any user defined parameters such as similarity thresholds, or a predefined number of clusters, and that it does not need any a priori knowledge on the data distribution itself. The main premise of our proposed method is the discovery of an intriguing behavior of chaining large number of samples based on a simple observation of the first neighbor of each data point. Since the groupings so obtained do not depend on any predefined similarity thresholds or other specific objectives, the algorithm may have the potential of discovering natural clusters in the data. The proposed method has low computational overhead, is extremely fast, handles large data and provides meaningful groupings of the data with high purity.

2. Related Work

Books have been written to guide through a myriad of clustering techniques [12]. The representative algorithms can largely be classified in three directions, centroid/partitioning algorithms (e.g., Kmeans, Affinity Propagation), hierarchical agglomerative/divisive methods and methods that view clustering as a graph partitioning problem (e.g., spectral clustering methods). For center-based clustering it is known that the Kmeans is sensitive to the selection of the initial K centroids. The affinity propagation algorithm [13] addresses this issue by viewing each sample as an

exemplar and then an efficient message parsing mechanism is employed until a group of good exemplars and their corresponding clusters emerge. Such partition-based methods are also commonly approached by choosing an objective function and then developing algorithms that approximately optimize that objective [29, 1, 32]. Spectral Clustering (SC) and its variants have gained popularity recently [36]. Most spectral clustering algorithms need to compute the full similarity graph Laplacian matrix and have quadratic complexities, thus severely restricting the scalability of spectral clustering to large data sets. Some approximate algorithms have been proposed [39, 23] to scale spectral methods. An important clustering direction has approached these spectral schemes by learning a sparse subspace where the data points are better separated, see Elhamifar and Vidal's Sparse Subspace Clustering (SSC) [11]. The aim is to reduce the ambiguity in the sense of distances in high dimensional feature spaces. Recently many methods approach estimating such subspaces by also using low-rank constraints, see Vidal et al. [35] and very recent work by Brbic and Kopriva [4].

In their remarkable classic work Jarvis & Patrick [17] bring forth the importance of shared nearest neighbors to define the similarity between points. The idea was rooted in the observation that two points are similar to the extent that their first k-neighbors match. Similarities so obtained are a better measure of distances between points than standard euclidean metrics. Using neighbors to define similarity between points has been used in partition-based, hierarchical and spectral clustering techniques [27, 5, 44].

Our paper closely relates to Hierarchical Agglomerative Clustering (HAC) methods, which have been studied extensively [30]. The popularity of hierarchical clustering stems from the fact that it produces a clustering tree that provides meaningful ways to interpret data at different levels of granularity. For this reason, there is a lot of interest in the community to both develop and study theoretical properties of hierarchical clustering methods. Some of the recent works establish guarantees on widely used hierarchical algorithms for a natural objective function [25, 9]. In agglomerative methods, each of the input sample points starts as a cluster. Then iteratively, pairs of similar clusters are merged according to some metric of similarity obtained via well studied linkage schemes. The most common linkage-based algorithms (single, average and complete-linkage) are often based on Kruskals minimum spanning tree algorithm [20]. The linkage methods merge two clusters based on the pairwise distances of the samples in them. The linkage schemes can be better approached by an objective function that links clusters based on minimizing the total within cluster variance e.g., Ward [37]. This approach generally produces better merges than the single or average linkage schemes. Dasgupta [10] recently proposed an objective function optimization on the similarity graph for hierarchical clustering to directly obtain an optimal tree. It initiated a line of work developing algorithms that explicitly optimize such objectives [31, 7, 25, 9].

Recently clustering is also used in jointly learning a non-linear embedding of samples. This could be approached with deep learning based methods e.g., employing auto encoders to optimize and learn the features by using an existing clustering method [40, 38, 15, 18]. These deep learning based approaches are primarily the feature representation learning schemes using an existing clustering method as a means of generating pseudo labels [6] or as an objective for training the neural network.

Almost all of the existing clustering methods operate directly on the distance values of the samples. This makes it hard for these methods to scale to large data as they need to have access to the full distances stored as floats. Apart from this, all of the current clustering methods need some form of supervision/expert knowledge, from requiring to specify the number of clusters to setting similarity thresholds or other algorithm specific hyper-parameters. Our proposal is a major shift from these methods in that we do not require any such prior knowledge and do not need to keep access to the full pairwise distance floats.

3. The Proposed Clustering Method

Historically, clustering methods obtain groupings of data by interpreting the direct distances between data points. Data that lies in high dimensional space has less informative measure of closeness in terms of these distances. Methods that aim at describing uniform volumes of the hyperspace may fail because source samples are hardly uniformly distributed within the target manifold. On the other hand, semantic relations (i.e., who is your best friend/ or friend of a friend) may be impervious to this as they rely on indirect relations rather than exact distances. Our proposal is intuitively related to such semantic relations for discovering the groupings in the data. We observed that the very first neighbor of each point is a sufficient statistic to discover linking chains in the data. Thus, merging the data into clusters can be achieved without the need to maintain a full distance matrix between all the data points. Also, using this approach, no thresholds or other hyper-parameters have to be set. We capture this observation in *the proposed clustering equation* and then describe the full algorithm in detail.

3.1. The Clustering Equation

Given the integer indices of the first neighbor of each data point, we directly define an adjacency link matrix

$$A(i,j) = \begin{cases} 1 & \text{if } j = \kappa_i^1 \text{ or } \kappa_j^1 = i \text{ or } \kappa_i^1 = \kappa_j^1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

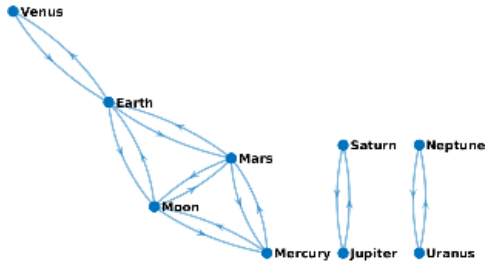
where κ_i^1 symbolizes the first neighbor of point i. The

		1st neighbor
1	Mercury	Moon
2	Venus	Earth
3	Earth	Mars
4	Moon	Mars
5	Mars	Moon
6	Jupiter	Saturn
7	Saturn	Jupiter
8	Uranus	Neptune
9	Neptune	Uranus

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	1	0	1	1	0	0	0	0
4	1	0	1	0	1	0	0	0	0
5	1	0	1	1	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0

(b)



(c)

Figure 1. The clustering equation walk-through: clustering the planets in our solar system. (a) Planets and their first neighbors. (b) Adjacency link matrix by Eq.1. (c) Directed graph using the adjacency matrix of (b). FINCH discovered 3 clusters shown as directed graph's connected components. Each planet is represented by first 15 attributes described in [https://nssdc.gsfc.nasa.gov/planetary/factsheet/].

adjacency matrix links each point i to its first neighbor via $j = \kappa_i^1$, enforces symmetry via $\kappa_j^1 = i$ and links points (i, j) that have the same neighbor with $\kappa_i^1 = \kappa_j^1$. Equation 1 returns a symmetric sparse matrix directly specifying a graph who's connected components are the clusters. One can see, in its striking simplicity, the clustering equation delivers clusters in the data without relying on any thresholds or further analysis. The connected components can be obtained efficiently from the adjacency matrix by using a directed or undirected graph $G = (V, E)$, where V is the set of nodes (the points to be clustered) and the edges E connect the nodes $A(i, j) = 1$. Intuitively the conditions in the Equation 1 are combining 1-nearest neighbor (1-nn) and shared nearest neighbour (SNN) graphs. Note, however, that the proposed clustering equation directly provides clusters without solving a graph segmentation problem. More precisely the adjacency matrix obtained from the proposed clustering equation has absolute links. We do not need to use any distance values as edge weights and require solving a graph partitioning. This is what makes it unique as by just using the integer indices of first neighbors, Equation 1 specifies and discover the semantic relations and directly delivers the clusters. Computing the adjacency matrix is also computationally very efficient since it can easily be obtained by simple sparse matrix multiplication and addition operation.

To understand the mechanics of Equation 1 and see how it chains large numbers of samples in the data, let's first look at a small tractable example of clustering the planets in our solar system. We can cluster the planets in the space of their known measurements e.g., mass, diameter, gravity, density, length of day, orbital period and orbital velocity etc. We describe each planet by the first 15 measurements taken from NASA's planetary fact sheet. The first neighbor of each planet in this space is the one with the minimum distance, obtained by computing pairwise euclidean distance. Figure 1 (a) shows the 9 samples (8 planets and moon) and their first neighbors. With this information one can now form the 9×9 adjacency matrix (see Fig. 1 (b)) according the Equation 1. An important observation is that not all first neighbors are symmetric, e.g., Earth is the first neighbor of Venus but Venus is not the first neighbor of Earth. This is also the basis of why these neighbors can form chains. Note how each of the adjacency conditions results in linking the planets. For instance, the adjacency condition $j = \kappa_i^1$ simply connects all planets to their first neighbors. The condition $\kappa_i^1 = \kappa_j^1$ connects Mercury-Mars and Earth-Moon together because they have same first neighbor. The condition $\kappa_j^1 = i$ forces symmetry and bridges the missing links in the chains, e.g., Earth is connected to Venus via this condition. Figure 1 (c) shows the directed graph of this adjacency matrix. Note how five out of nine planets are chained together directly while symmetric neighbors formed two separate clusters. This short walk-through explains the mechanics of Equation 1 which has discovered three clusters. Interestingly astronomers also distinguish these planets into three groups at a fine level, rocky planets (Mercury, Venus, Earth, Moon, and Mars) in one group, gas planets (Jupiter, Saturn) being similar in terms of larger size with metallic cores, and ice giants (Uranus, Neptune) are grouped together because of similar atmospheres with rocky cores.

3.2. Proposed Hierarchical Clustering

The question whether the discovered clusters are indeed the true groupings in the data has no obvious answer. It is because the notion of clusters one considers true are subjective opinions of the observer. The problem of finding ground-truth clustering has been well studied in Balcan *et al.* [2], where they show that having a list of partitions or a hierarchy instead of a single flat partition should be preferred. In such a set of groupings, they show that single or average-linkage algorithms are known to provably recover the ground-truth clustering under some properties of a similarity function.

Equation 1 delivers a flat partition of data into some automatically discovered clusters. Following it up recursively in an agglomerative fashion may provide further successive groupings of this partition capturing the underlying data structure at different level of granularities. Because Equation 1 may chain large numbers of samples quickly, we show that in only a few recursions a meaningful small set of partitions is obtained with a very high likelihood of recovering

Algorithm 1 Proposed Algorithm

Input: Sample set $S = \{1, 2, \cdots, N\}$, $S \in \mathbb{R}^{N \times d}$, where N is total number of samples and each sample point is represented by d attributes or feature dimensions.

Output: Set of Partitions $\mathcal{L} = \{\Gamma_1, \Gamma_2, \cdots, \Gamma_L\}$ where each partition $\Gamma_i = \{C_1, C_2, \cdots, C_{\Gamma_i} | C_{\Gamma_i} > C_{\Gamma_i+1} \forall i \in \mathcal{L}\}$ is a valid clustering of S.

The FINCH Algorithm:

Compute first neighbors integer vector $\kappa^1 \in \mathbb{R}^{N \times 1}$ via exact distance or fast approximate nearest neighbor methods.

Given κ^1 get first partition Γ_1 with C_{Γ_1} clusters via Equation 1. C_{Γ_1} is the total number of clusters in partition Γ_1.

while there are at least two clusters in Γ_i **do**

  Given input data S and its partition Γ_i, compute cluster means (average of all data vectors in that cluster). Prepare new data matrix $M = \{1, 2, \cdots, C_{\Gamma_i}\}$, where $\mathbb{M}^{C_{\Gamma_i} \times d}$.

  Compute first neighbors integer vector $\kappa^1 \in \mathbb{R}^{C_{\Gamma_i} \times 1}$ of points in M.

  Given κ^1 get partition Γ_M of Γ_i via Equation 1, where $\Gamma_M \supseteq \Gamma_i$

  if Γ_M has one cluster **then**

    break

  else

    Update cluster labels in $\Gamma_i : \Gamma_M \rightarrow \Gamma_i$

  end if

end while

the exact ground-truth clustering or a partition very close to it. Since by just using the **F**irst **I**nteger **N**eighbor indices we can produce a **C**lustering **H**ierarchy, we term our algorithm as (FINCH).

The FINCH Algorithm. The main flow of the proposed algorithm is straightforward. After computing the first partition, we want to merge these clusters recursively. For this, Equation 1 requires the first neighbor of each of these clusters. Finding these neighbors requires computing distances between clusters. This is also related to all the linkage methods, e.g., average-linkage in hierarchical agglomerative clustering use the average of pairwise distances between all points in the two clusters. Following this, however, is computationally very expensive with a complexity of $\mathcal{O}(N^2 log(N))$. It thus can not easily scale to millions of samples, and it requires to keep the full distance matrix in the memory. With our method, for large scale data, we do not need to compute the full pairwise distance matrix as we can obtain first neighbors via fast approximate nearest neighbor methods (such as k-d tree). We, instead, average the data samples in each cluster and use these mean vectors to compute the first neighbor. This simplifies the computation and keeps the complexity to $\mathcal{O}(Nlog(N))$ as we merge these clusters at each recursion. In contrast to standard HAC algorithms, our clustering equation directly provides a meaningful partition of the data at each iteration. It provides a hierarchical structure as a small set of partitions where each successive partition is a superset of the preceding partitions. The resulting set of partitions provides a fine to coarse view on the discovered groupings of data. The complete algorithmic steps are described in Algorithm 1.

Algorithm 2 Required Number of Clusters Mode

Input: Sample set $S = \{1, 2, \cdots, N\}$, $S \in \mathbb{R}^{N \times d}$ and a partition Γ_i from the output of Algorithm 1.

Output: Partition Γ_r with required number of clusters.

for steps = C_{Γ_i} - C_{Γ_r} **do**

  Given input data S and its partition Γ_i, compute cluster means (average of all data vectors in that cluster). Prepare new data matrix $M = \{1, 2, \cdots, C_{\Gamma_i}\}$, where $\mathbb{M}^{C_{\Gamma_i} \times d}$.

  Compute first neighbors integer vector $\kappa^1 \in \mathbb{R}^{C_{\Gamma_i} \times 1}$ of points in M.

  Given κ^1 compute Adjacency Matrix of Equation 1

  $\forall A(i, j) = 1$ keep one symmetric link $A(i, j)$ which has the minimum distance $d(i, j)$ and set all others to zero.

  Update cluster labels in Γ_i: Merge corresponding (i, j) clusters in Γ_i

end for

The quality of the outlined procedure is demonstrated in the experimental section on diverse data of different sizes. We show that it results in partitions that are very close to the ground-truth clusters. For some applications, it is often a requirement to obtain a specific number of clusters of the data. Since our algorithm returns a hierarchy tree, we can use simple mechanisms to refine a close partition by one merge at a time to provide required clusters as well. However, the required number of clusters are obviously upper-bounded by the size of the first partition, FINCH returns. For completeness, we outline a simple procedure in Algorithm 2.

For better clarity, we demonstrate the quality of the outlined FINCH algorithm in Fig. 2 using the classic 2D Gestalt [41] and Aggregation [14] data that represent well understood clustering problems and provide a good qualitative look into the performance of clustering methods. After running FINCH Algo 1 we obtained a set of clustering partitions. Here, to evaluate FINCH on the true clusters, we can take a larger partition than the true clusters and refine it with Algo 2. For example, on Gestalt data with 399 2D points, Algo 1 provides a total of 4 partitions with $\{91, 23, 5$ and $2\}$ clusters. We use Algo 2 on the 23 cluster partition to evaluate at ground truth 6 clusters. Clearly, we can observe

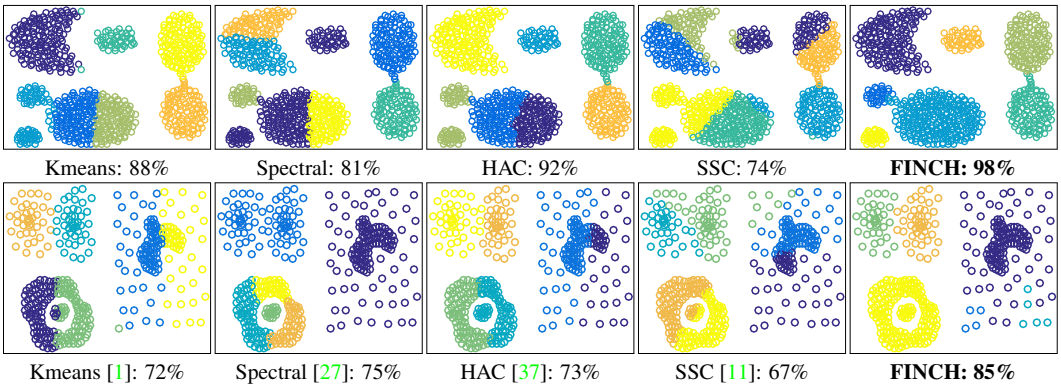


Figure 2. Visualization of Aggregation [14] 7 clusters (top) and Gestalt [41] 6 clusters (bottom) with NMI score for each method.

that FINCH maintains the merges quite well and clusters these data better than the considered baselines.

4. Experiments

We demonstrate FINCH on challenging datasets which cover domains such as biological data, text, digits, faces and objects. We first introduce the datasets and clustering metric, followed by a thorough comparison of the proposed method to algorithms that estimate the clusters automatically using hyper-parameter settings, and also to algorithms that require #clusters as input. We also compare FINCH with state-of-the-art deep clustering methods in section 4.2.

Datasets. The datasets are summarized in Table 1. The dimensionality of the data in the different datasets varies from 77 to 4096. `Mice Protein` [16] consists of the expression levels of 77 proteins measured in eight classes of control and trisomic genotypes, available from [32]. `REUTERS` [22] consists of 800000 Reuters newswire articles. We use a sampled subset of 10000 articles available from [38], categorized in four classes. The TF-IDF features are of 2000 dimensions. `STL-10` [8] is an image recognition dataset for unsupervised learning, consisting of 10 classes with 1300 examples each. `BBTs01` (season 1, episodes 1 to 6) and `BFs05` (season 5, episodes 1 to 6) are challenging video face identification/clustering datasets. They are sitcoms with small cast list, for BBTs01: 5 main casts, and BFs05: 6 main casts. Data for BBTs01 and BFs05 are provided by [3]. `MNIST` [21]: we use three variants of MNIST handwritten digits, they consist of: 10K testset (`MNIST_10k`), 70K (train+test) (`MNIST_70k`), and 8.1M [24] (`MNIST_8M`) samples categorized into 10 classes. We use CNN features for STL-10, BBTs01, BFs05, and both CNN features and pixels for MNIST datasets. More details on the features used and datasets settings can be found in the supplimentary .

Evaluation Metrics. We use the most common clustering evaluation metric, Normalized Mutual Information (NMI), and the unsupervised clustering accuracy (ACC) as the metric to evaluate the quality of clustering. ACC is also widely used [38, 15, 18] and computed as $\max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{l_i = m(c_i)\}}{n}$, where l_i is the ground truth label, c_i is the cluster assignment obtained by the method, and m ranges in the all possible one-to-one mappings between clusters and labels.

Baselines. We compare FINCH to existing clustering methods (both classic and recent proposals) that covers the whole spectrum of representative clustering directions. We include 11 baselines categorized in two variants of clustering algorithms: (1) algorithms that estimate the number of clusters automatically - given some input hyper-parameter/threshold settings. These algorithms include Affinity Propagation (AP) [13], Jarvis-Patrick (JP) [17], Rank-Order (RO) [44], and the recently proposed Robust Continuous Clustering (RCC) [32]; and (2) algorithms that require the number of clusters as input. These algorithms are: *Kmeans++* (Kmeans) [1], Birch (BR) [42], Spectral (SC) [27], Sparse Subspace Clustering (SSC) [11], Hierarchical Agglomerative Clustering (HAC_Ward) with ward linkage [37], and HAC with average linkage (HAC_Avg), and very recent method Multi-view Low-rank Sparse Subspace Clustering (MV-LRSSC) [4]. The parameter settings for the baselines are provided in the *supplimentary*.

4.1. Comparison with baselines

Comparison on Small-scale datasets: In this section, we consider clustering datasets upto 70k samples. Considering the size of BBTs01 ($\sim$199k) samples, the full distance matrix takes up approximately 148 GB RAM. The memory consumption of different algorithms is not linear rather exponential, and the number of samples and feature dimensions parameters negatively influence their time efficiency and computational cost. For this reason, we separate algorithms that need to store quadratic memory usage. The algorithms that need to compute the full distance matrix are evaluated in this section: small scale ($\leq$70k: $\sim$36.5 GB), while large scale ($\geq$199k: $\sim$148 GB) are evaluated separately.

In Table 2, we compare the performance of FINCH with the current clustering methods. Results on datasets: MICE Protein, REUTERS, STL-10, MNIST_10k and MNIST_70k are reported in Table 2 using NMI-measure. We compare FINCH against the algorithms that requires the number of

	Mice Protein	REUTERS	STL-10	BBTs01	BFs05	MNIST		
Type	Biological	Text	Objects	Faces		Digits		
#C	8	4	10	5	6	10		
#S	1077	10k	13k	199346	206254	10k	70k	8.1M
Dim.	77	2000	2048	2048	2048	4096/784	4096/784	256/784
LC/SC (%)	13.88/9.72	43.12/8.14	10/10	33.17/0.63	39.98/0.61	11.35/8.92	11.25/9.01	11.23/9.03

Table 1. Datasets used in experiments. #S denotes the number of samples, #C denotes the true number classes/clusters, and largest class (LC) / smallest class (SC) is the class balance percent of the given data.

	NMI Scores													
	Algorithms that estimate #C automatically					@FINCH estimated #C								
Dataset	**FINCH**	AP	JP	RO	RCC	Kmeans	BR	SC	HAC_Ward	HAC_Avg	SSC	MV-LRSSC	True #C	#S
Mice Protein	**51.64**	59.10	55.99	1.75	**65.92**	42.66	40.39	55.13	51.35	37.65	41.94	51.31		
Estim. #C	**8**	67	30	2	38				8				8	1077
REUTERS	**44.88**	36.23	22.97	36.76	28.32	41.75	38.77	7.97	38.40	12.38	3.19	41.27		
Estim. #C	**4**	1073	1656	9937	358				4				4	10k
STL-10	**85.05**	57.18	51.70	33.37	81.56	85.59	80.9	72.62	80.9	52.57	81.25	74.44		
Estim. #C	**10**	589	4780	4358	14				10				10	13k
MNIST_10k	**97.55**	69.97	35.97	49.87	77.74	81.92	80.78	97.43	89.05	63.86	96.63	93.67		
Estim. #C	**10**	116	513	9950	149				10				10	10k
MNIST_70k	**98.84**	–	24.20	4.01	86.59	81.02	84.50	98.77	87.61	47.08	–	–		
Estim. #C	**10**	–	5722	531	120				10				10	70k

Table 2. Small-scale clustering results of FINCH with nearest neighbors obtained using exact distances. We compare FINCH against algorithms that estimates the clusters automatically - given input hyper-parameters/thresholds, and the algorithms that requires the #clusters as input. For algorithms that require the number of clusters as input, we use the #clusters estimated by FINCH. – denotes OUT_OF_MEMORY.

cluster as input: Kmeans, BR, SC, HAC, SSC and MV-LRSSC. To have a fair time comparison with these algorithms we also compute the full pairwise distance matrix for obtaining the first neighbours. To demonstrate the quality of merges FINCH made, we use the FINCH estimated clusters as input to these algorithms. On these datasets, FINCH has estimated the ground truth clusters as one of its partition, see Fig 3 and discussion in section 5.

For comparison with algorithms that provides a fixed flat partition given some hyperparamters (AP, JP, RO and RCC), we can not directly compare FINCH as it provides a hierarchy. Here we follow the previous approaches [28, 32, 33] that tend to compare on the basis of NMI measure only, and not on the basis of estimated number of clusters. In Table 2, following the same procedure, we simply evaluate all the algorithms at the respective author's best parameter setup for each method, and report their results attained. We observe that not only FINCH finds a meaningful partition of the data it also consistently achieves high performance on most of these datasets.

Comparison on Large-scale datasets: As FINCH only requires the first neighbor indices, for large scale datasets we obtain the first nearest neighbor using the randomized k-d tree algorithm from [26], thus avoiding the expensive $\mathcal{O}(N^2)$ distance matrix computation cost, and quadratic memory storage. For example, computing the full distance matrix for single precision MNIST_8M requires 244416.3 GB RAM.

Among all our considered baselines, only Kmeans and RCC are able to scale. We compare FINCH against Kmeans and RCC on BBTs01 and BFs05 datasets. On the million scale (8.1 million samples) MNIST_8M datasets, we were only able to run Kmeans.

Dataset	NMI		
	FINCH	RCC	Kmeans
BBTs01	**89.79**	2.56	71.82
Estim. #C	6	7	6
BBTs01 (@True #C=5)	**91.57**	–	83.39
BFs05	**82.46**	46.70	71.85
Estim. #C	7	523	7
BFs05 (@True #C=6)	**83.64**	–	76.15

Table 3. BBTs01 and BFs05 (∼200k).

Dataset	NMI	
	FINCH	Kmeans
MNIST_8M_CNN (@Estim. #C=13)	**96.55**	93.33
MNIST_8M_CNN (@True #C=10)	**99.54**	97.39
MNIST_8M_PIXELS (@Estim. #C=11)	**46.49**	40.17
MNIST_8M_PIXELS (@True #C=10)	**63.84**	37.26

Table 4. MNIST_8M (8.1M).

For BBTs01 and BFs05, there exists a huge line of work, from exploiting video-level constraints [34], to dynamic clustering constraints via MRF [43], and the most recent link-based clustering [19]. In contrast to these works, we use features from a pre-trained model on other datasets without any data specific transfer or considering any other video-level constraints for clustering. FINCH performs significantly better in comparison with the previously published methods on these datasets. These comparison results are available in the *supplimentary*. Results in Table 3 and run-time in Table 5 show that with approximate nearest neighbors, FINCH

Dataset	#S	Dimen.	**FINCH**	Kmeans	SC	HAC_Ward	HAC_Avg	AP	JP	RO	BR	RCC	SSC	MV-LRSSC
Mice Protein	1077	77	**37ms**	115ms	220ms	40ms	668ms	700ms	00:01	00:02	90ms	84ms	00:08	00:02
REUTERS	10k	2000	**00:05**	00:18	05:54	00:31	00:43	01:53	40:25	00:14	01:32	37:25	01:27:36	52:53
STL-10	13k	2048	**00:03**	00:19	08:03	00:42	01:10	02:42	57:49	00:07	02:25	15:11	02:41:14	02:04:52
MNIST_10k	10k	4096	**00:10**	00:19	02:39	01:05	01:31	02:23	44:20	00:12	03:06	13:41	01:35:25	38:42
MNIST_70k	70k	4096	**00:54**	02:19	58:45	29:28	30:17	–	60:09:17	05:22	02:20:44	05:53:43	–	–
BBTs01	199346	2048	**01:06**	02:17	–	–	–	–	–	–	–	00:38:11	–	–
BFs05	206254	2048	**01:09**	01:33	–	–	–	–	–	–	–	03:28:04	–	–
MNIST_8M	8.1M	256	**18:23**	56:41	–	–	–	–	–	–	–	–	–	–
Framework			Matlab	Python	Python	Matlab	Matlab	Python	Matlab	C++	Python	Python	Matlab	Matlab

Table 5. Run-time comparison of FINCH with Kmeans, SC, HAC, AP, JP, RO, BR, RCC SSC, and MV-LRSSC. We report the run time in HH:MM:SS and MM:SS. – denotes OUT_OF_MEMORY.

achieves the best run-time of the three methods and the best performance as well. A similar behavior can be observed in Table 4 for very large scale MNIST_8M datasets.

4.2. Deep Clustering: Unsupervised Learning

Many recent methods propose to learn feature representations using a given clustering algorithm as objective, or as a means of providing weak labels/pseudo labels for training a deep model (usually an auto-encoder) [40, 38, 15, 18, 6]. FINCH lends itself naturally to this task since it provides a hierarchy of partitions containing automatically discovered natural groupings of data and one need not specify a user defined number of clusters to train the network. To demonstrate this and to be able to compare with deep clustering methods we follow a similar approach, as used recently in [6], of using cluster labels to train a small network for learning an unsupervised feature embedding. We demonstrate this unsupervised learning on `MNIST_70K_PIXELS` (28x28 =784-dim), `REUTERS-10k` TF-IDF (2000-dim) and `STL-10` ResNet50 (2048-dim) features as input. Note that we use the same features that were used in previous methods to train their embedding, see Jiang *et al*. [18].

We train a simple MLP with two hidden layers for this task in classification mode. Among the FINCH returned partitions the partition obtained at the first pass of the Equation 1 or the one after it contains the largest number of clusters and also the purest since follow on recursions are merging these into smaller clusters. We use the estimated clusters of the second partition as the pseudo labels to train our network with softmax. We approach the training in steps, initializing training with the FINCH labels and re-clustering the last layer embedding after 20 epochs to update the labels. At each label update step we divide the learning rate by 10 and train for 60-100 epochs. For all experiments our network structure is fixed with the two hidden layers set to 512 neurons each. We train the network with minibatch SGD with batch size of 256 and initial learning rate of 0.01. The last layer 512-dimensional embedding is trained this way and used to report clustering performance at the ground truth clusters. As seen in Table 6 FINCH has effectively trained an unsupervised embedding that clusters better in this learned space. Interestingly, on STL-10 for which we have ResNet50 features, FINCH directly clusters these base features with better accuracy as the compared methods achieve after training a deep model. After our FINCH driven deep clustering we are able to improve clustering performance on STL-10 to $\sim 95\%$ improving by almost 10% on the existing state-of-the-art deep clustering methods.

Method	Accuracy (ACC %)		
	MNIST_70k_PIXELS	REUTERS-10K	STL-10
GMM [18]	53.73	54.72	72.44
AE+GMM [18]	82.18	70.13	79.83
VAE+GMM [18]	72.94	69.56	78.86
DEC [38]	84.30	72.17	80.62
IDEC [15]	88.06	76.05	-
VaDE [18]	**94.46**	79.83	84.45
FINCH on base features	74.00	66.14	85.28
DeepClustering With FINCH	91.89	**81.46**	**95.24**

Table 6. Unsupervised Deep Clustering with FINCH: Comparison with state-of-the-art deep clustering methods at true clusters.

4.3. Computational Advantage

In Table 5, we report the run-time of each algorithm for all the datasets. The corresponding timing is the complete time for the algorithm execution, that includes computing the pairwise distance between the samples, or obtaining the nearest neighbors using kd-tree, and the running time of each algorithm. We can observe that, FINCH achieves a remarkable time efficiency, and is not dominated by the number of samples, and/or the feature dimensions. Apart from time efficiency, FINCH is very memory efficient, as it only requires to keep the integer $(N \times 1)$ first neighbor indices and the data. The performance is measured on a workstation with an AMD Ryzen Threadripper 1950X 16-core processor with 192 (128+64 swap) GB RAM. For large scale datasets with more than 70k samples, most of the algorithms break, or demands for more than 192 GB RAM. FINCH memory requirement is, therefore, $\mathcal{O}(N)$ vs $\mathcal{O}(N^2)$. The computational complexity of FINCH is $\mathcal{O}(Nlog(N))$, whereas spectral methods are $\mathcal{O}(N^3)$ and hierarchical agglomerative linkage-based methods are $\mathcal{O}(N^2log(N))$.

5. Discussion

We have extensively evaluated FINCH on a number of different data (image pixels, biological measurements, text

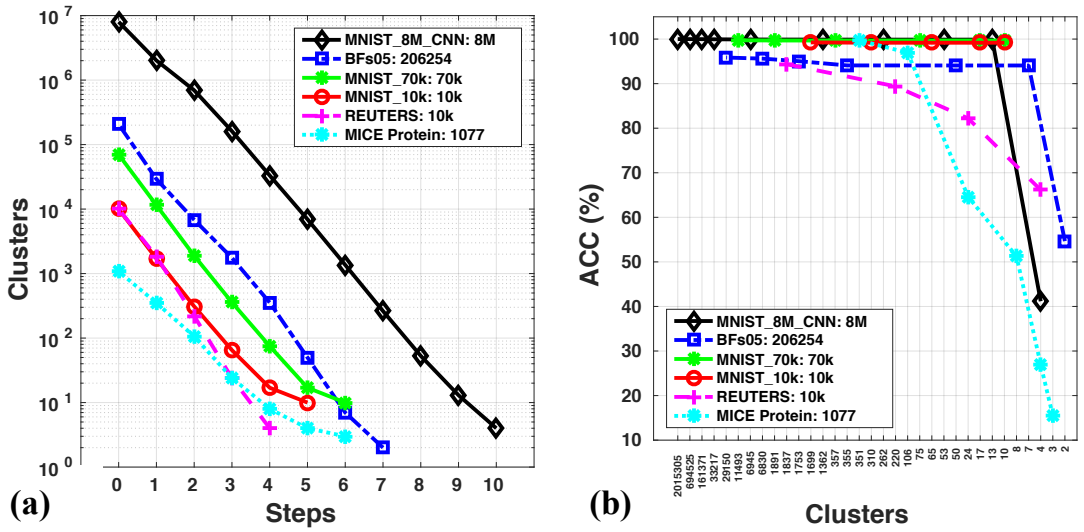


Figure 3. FINCH steps/partitions and clustering quality. (a) Algorithm convergence: clusters vs algorithm steps, each step produces a partition of data. (b) Quality of merges: number of cluster produced at each step and their corresponding purity/accuracy

frequency and CNN features) represented in 77 to 4096 dimensional feature space. Interestingly many existing popular clustering methods do not perform well even in the cases where the data has well separated distinct clusters in the feature space. The recursion scheme in our algorithm converges at an exponential rate in very few steps providing a valid partition of the data at each step. In Figure 3 (a) we plot the clusters obtained at each step of the FINCH algorithm. One can see for the different data sizes (1077 samples to 8.1 million samples), our algorithm converges in 4-10 steps providing a valid partition at each. The accuracy of the formed clusters at each partition is depicted in Figure 3 (b). One can assess the quality of discovered clusters and the successive merges by the fact that it maintains the accuracy quite well for very large merges through each step. Corresponding to Figure 3 (a) the *x-axis* of Figure 3 (b) depicts #clusters obtained at each step of the algorithm for all of the plotted datasets. For example, for the smallest dataset with 1077 samples, FINCH delivers a total of 6 partitions in 6 steps of the run. From 1077 samples it comes down to 351 clusters in the first pass of Equation 1 and then to 106, 24, 8, 4 and 3 clusters in the successive recursions, discovering the ground-truth clustering of 8 clusters at step 4 of the algorithm. One can interpret the corresponding cluster purity and the number of clusters at each step of FINCH for the other datasets in Figure 3 (b). In direct contrast to HAC linkage schemes which require $N-1$ steps to converge for N samples, FINCH convergence (number of steps) does not depend on N and is governed by Equation 1. It is very interesting to study this behavior and finding some theoretical bounds explaining this. An interesting observation are the results on the MNIST_10k and MNIST_70k datasets in Table 2. Since these features have been learned on the data using the ground-truth labels, these are already well separated 10 clusters in the feature space. Despite this, none of the baseline algorithms (even the ones that require to specify the number of clusters) performs as accurately. FINCH stops its recursive merges at step 5 (for MNIST_10k) and step 6 (for MNIST_70k) providing the exact 10 clusters with above 99% accuracy. Note that this is the same classification accuracy as the trained CNN model provides on these features. Since the ground-truth clusters are well separated, Algorithm 1 exactly stops here because another pass of this 10-cluster partition merge them to 1 cluster. This shows that FINCH can recover well the global structure in the data. We also include the number of steps and corresponding clusters, along with the accuracy, for each dataset in the *supplimentary*. Note that, because of the definition of Equation 1, FINCH can not discover singletons (clusters with 1 sample). This is because we link each sample to its first neighbor without considering the distance between them. Singletons, therefore will always be paired to their nearest sample point. This sets the limit of smallest cluster with size 2 for FINCH.

Conclusively, we have presented an algorithm that shifts from the prevalent body of clustering methods that need to keep the pairwise distance matrix and require user defined hyper-parameters. It offers a unique and simple solution in the form of the clustering equation and an effective agglomerative merging procedure. The advantage it brings, in being fully parameter-free and easily scalable to *large data* at a minimal computational expense, may prove useful for many applications. We have demonstrated one such application of unsupervised feature representation learning. Automatically discovering meaningful clusters in such a manner is needed in many areas of sciences where nothing is known about the structure of data. For example, it may have very exciting applications from discovering exotic particles to stars/galaxies based on their spectra.

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *ACM-SIAM*. Society for Industrial and Applied Mathematics, 2007.

[2] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *ACM STOC*, 2008.

[3] Martin Bäuml, Makarand Tapaswi, and Rainer Stiefelhagen. Semi-supervised Learning with Constraints for Person Identification in Multimedia Data. In *CVPR*, 2013.

[4] Maria Brbić and Ivica Kopriva. Multi-view low-rank sparse subspace clustering. *Pattern Recognition*, 2018.

[5] Sébastien Bubeck and Ulrike von Luxburg. Nearest neighbor clustering: A baseline method for consistent clustering with arbitrary objective functions. *JMLR*, 2009.

[6] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[7] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *ACM SIAM*, 2017.

[8] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[9] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *NIPS*, 2017.

[10] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *ACM STOC*, 2016.

[11] Ehsan Elhamifar and Rene Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *PAMI*, 2013.

[12] Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster Analysis, 5th Edition*. Wiley-Blackwell, 2011.

[13] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 2007.

[14] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM TKDD*, 2007.

[15] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, 2017.

[16] Clara Higuera, Katheleen J Gardiner, and Krzysztof J Cios. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PloS one*, 2015.

[17] Raymond Austin Jarvis and Edward A Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on computers*, 1973.

[18] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *IJCAI*, 2017.

[19] SouYoung Jin, Hang Su, Chris Stauffer, and Erik Learned-Miller. End-to-end Face Detection and Cast Grouping in Movies using ErdsRnyi Clustering. In *ICCV*, 2017.

[20] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *American Mathematical society*, 1956.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 1998.

[22] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[23] Mu Li, Xiao-Chen Lian, James T Kwok, and Bao-Liang Lu. Time and space efficient spectral clustering via column sampling. In *CVPR*, 2011.

[24] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, 2007.

[25] Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *NIPS*, 2017.

[26] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *PAMI*, 2014.

[27] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, 2002.

[28] Charles Otto, Dayong Wang, and Anil K Jain. Clustering millions of faces by identity. *PAMI*, 2018.

[29] Adrian E Raftery and Nema Dean. Variable selection for model-based clustering. *American Statistical Association*, 2006.

[30] Chandan K Reddy and Bhanukiran Vinzamuri. A survey of partitional and hierarchical clustering algorithms. *Data Clustering: Algorithms and Applications*, 2013.

[31] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *NIPS*, 2016.

[32] Sohil Atul Shah and Vladlen Koltun. Robust continuous clustering. *PNAS*, 2017.

[33] Sohil Atul Shah and Vladlen Koltun. Deep continuous clustering. *arXiv:1803.01449*, 2018.

[34] Makarand Tapaswi, Omkar M Parkhi, Esa Rahtu, Eric Sommerlade, Rainer Stiefelhagen, and Andrew Zisserman. Total Cluster: A Person Agnostic Clustering Method for Broadcast Videos. In *ICVGIP*, 2014.

[35] René Vidal and Paolo Favaro. Low rank subspace clustering (lrsc). *Pattern Recognition Letters*, 2014.

[36] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 2007.

[37] Joe H. Ward Jr. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 1963.

[38] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, 2016.

[39] Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *ACM SIGKDD*, 2009.

[40] Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *CVPR*, 2016.

[41] Charles T Zahn. Graph theoretical methods for detecting and describing gestalt clusters. *IEEE TOC*, 1970.

[42] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, 1996.

[43] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Joint face representation adaptation and clustering in videos. In *ECCV*, 2016.

[44] Chunhui Zhu, Fang Wen, and Jian Sun. A rank-order distance based clustering algorithm for face tagging. In *CVPR*, 2011.