Grouping Points by Shared Subspaces for Effective Subspace Clustering

Ye Zhu[a,*], Kai Ming Ting[b], Mark J. Carman[c]

[a] *School of Information Technology, Deakin University, Victoria, Australia 3125*
[b] *School of Engineering and Information Technology, Federation University, Victoria, Australia 3842*
[c] *Faculty of Information Technology, Monash University, Victoria, Australia 3800*

Abstract

Clusters may exist in different subspaces of a multidimensional dataset. Traditional full-space clustering algorithms have difficulty in identifying these clusters. Various subspace clustering algorithms have used different subspace search strategies. They require clustering to assess whether cluster(s) exist in a subspace. In addition, all of them perform clustering by measuring similarity between points in the given feature space. As a result, the subspace selection and clustering processes are tightly coupled. In this paper, we propose a new subspace clustering framework named *CSSub* (Clustering by Shared Subspaces). It enables neighbouring core points to be clustered based on the number of subspaces they share. It explicitly splits candidate subspace selection and clustering into two separate processes, enabling different types of cluster definitions to be employed easily. Through extensive experiments on synthetic and real-world datasets, we demonstrate that *CSSub* discovers non-redundant subspace clusters with arbitrary shapes in noisy data; and it significantly outperforms existing state-of-the-art subspace clustering algorithms.

Keywords: Subspace clustering, Shared subspaces, Density-based clustering.

1. Introduction

Clustering detects groups of similar points while separating dissimilar points into different groups [1]. Many clustering algorithms have been studied in the research community over the past few decades.

Clustering is a fundamental data analysis technique for data mining and knowledge discovery and has been applied to various fields such as engineering, medical and biological sciences, social sciences and economics.

Traditional "full-space clustering" algorithms become ineffective when clusters exist in different subspaces[1] as many irrelevant attributes may cause similarity measurements used by the algorithms

*Corresponding author
Email addresses: ye.zhu@ieee.org (Ye Zhu), kaiming.ting@federation.edu.au (Kai Ming Ting), mark.carman@monash.edu (Mark J. Carman)

[1] By subspace we intend a linear subspace over a subset of the original dimensions. Moreover, we say that a "cluster exists in a subspace" if the cluster is defined as a region within the subspace, and thus dimensions not included in the

to become unreliable [1]. To tackle this problem, the similarity of points should be assessed within subspaces (over a subset of relevant attributes).

Subspace clustering aims to discover clusters which exist in different subspaces [2]. It works by combining subspace search and clustering procedures. The number of possible axis-parallel subspaces is exponential in the dimensionality and the number of possible arbitrarily oriented (non axis-parallel) subspaces is infinite. To deal with the huge number of possible subspaces, subspace clustering algorithms usually rely on heuristics for subspace search, e.g., top-down search [3] and bottom-up search [4]. All these existing subspace clustering algorithms perform clustering by measuring the similarity between points in the given feature space, and the subspace selection and clustering processes are tightly coupled.

In this paper, we focus on clustering in axis-parallel subspaces. We contribute a new subspace clustering framework, named *CSSub* (Clustering by Shared Subspaces) [2], which has the following three unique features:

1. *CSSub* groups points by their shared subspaces. It performs clustering by measuring the similarity between points based on the number of subspaces they share. This enables *CSSub* to detect non-redundant/non-overlapping subspace clusters directly by running a clustering method only once. In contrast, many existing subspace clustering algorithms need to run a chosen clustering method for each subspace; and this must be repeated for an exponentially large number of subspaces. As a consequence, it produces many redundant subspace clusters.

2. *CSSub* decouples the candidate subspace selection process from the clustering process. By explicitly splitting them into independent processes, it enables candidate subspaces to be selected independent of the clustering process—eliminating the need to repeat the clustering step a large number of times. In contrast, many existing subspace clustering algorithms which have tightly-coupled processes must rely on an anti-monotonicity property to prune the search space.

3. The decoupling approach has an added advantage that allows different types of cluster definitions to be employed easily. The time cost of the entire process is dominated by the subspace scoring function. We show that by changing the cluster definition such that subspaces can be evaluated with a linear scoring function, enables the runtime of *CSSub* to be reduced from quadratic time to linear time. A similar change is difficult, if not impossible, for existing algorithms because of the tightly coupled processes.

We present an extensive empirical evaluation on synthetic and real-world datasets to demonstrate

subspace are irrelevant and can be considered noise with respect to the cluster.

[2]The source code of *CSSub* can be obtained at `https://sourceforge.net/projects/subspace-clustering/`.

the effectiveness of *CSSub*. The experiments show that *CSSub* discovers subspace clusters with arbitrary shapes in noisy data; and it significantly outperforms existing state-of-the-art subspace clustering algorithms. In addition, *CSSub* has only one parameter k (the number of clusters) which needs to be manually tuned, other parameters can be automatically set based on a heuristic method.

The rest of this paper is organised as follows. We provide an overview of subspace clustering algorithms and related work in Section 2. Section 3 discusses key weaknesses of existing bottom-up subspace clustering algorithms. Section 4 details the subspace clustering framework based on the shared subspaces and presents a density-based approach for subspace scoring. Section 5 presents the algorithms for *CSSub*. In Section 6, we empirically evaluate the performance of the proposed algorithms on different datasets. Discussion and the conclusions are provided in the last two sections.

2. Related Work

The key task of clustering in subspaces is to develop appropriate subspace search heuristics [2]. There are two basic techniques for subspace search, namely top-down search [3] and bottom-up search [4]. Different subspace clustering algorithms have been proposed based on these two search directions [5, 6, 7, 8, 9]. Search strategies can be further subdivided into systematic and non-systematic search techniques as discussed below.

2.1. Systematic Subspace Search

Systematic search strategies for subspace clustering can be top down [3] or bottom up [4], as shown in Figure 1. Searching all possible subspaces generates $2^d - 2$ candidates (where d is the number of distinct attributes). To avoid having to perform an exhaustive search, the two systematic search strategies utilise certain criteria to prune the search space.

2.1.1. Bottom-up Methods

The bottom-up subspace search strategy uses the *Apriori* algorithm [5]. It starts from all one-dimensional subspaces and then searches for higher dimensional subspaces progressively. This strategy relies on the anti-monotonicity property to prune the search space: if a candidate subspace in a lower dimensional space has no clusters or is filtered out according to some other criteria, then its projection onto a higher dimensional space will not be traversed.[3]

[3]To be precise, the anti-monotonicity property is a property of density-based clustering approaches only, where points can be considered either "high density and therefore within a cluster" or "low density and therefore noise". Since density monotonically reduces with increased dimension, if a fixed density threshold is used to define points as high density then the property will be anti-monotone (in the set of attributes), i.e., a low-density point in a subspace with a set of attributes is still a low-density point in a higher dimensional subspace which contains these attributes.

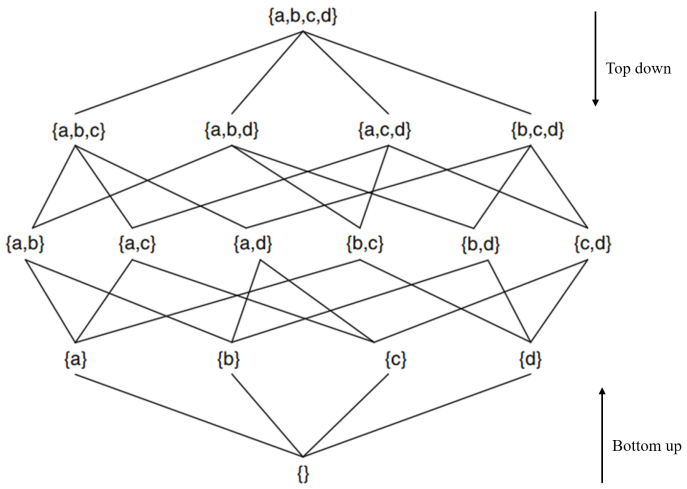


Figure 1: The lattice of candidate subspaces from attributes $\{a, b, c, d\}$.

CLIQUE [4] uses a grid-based structure to investigate dense units at $k-1$ dimensional subspace to produce the possible set of k-dimensional units that might contain dense units. MAFIA [10] is an extension of CLIQUE that uses a variable-width grid-based density estimator in each subspace in order to improve efficiency and quality. ENCLUS [11] is similar to CLIQUE, but it uses entropy rather than density to measure the "clusterability" of a subspace, i.e., subspaces having lower entropy normally contain more meaningful clusters. ENCLUS uses the same bottom-up fashion of CLIQUE for finding subspace clusters. However, the grid-based algorithm cannot be applied on high-dimensional datasets as the number of cells in the grid grows exponentially with the dimensionality of data.

SSCS [12] is based on the monotonicity property of density-based subspace clustering to tackle the exponential subspace search problem. Instead of performing step-by-step scheme of the *Apriori* algorithm, it proposes a best-first style such that it directly steers to high dimensional subspaces from low dimensional subspaces with successive information gathering about dense subspace regions identified by hyper-rectangles. It only outputs the most interesting clusters in a few high-dimensional subspaces for non-redundant clustering.

P3C [13] is a grid-based algorithm relying on statistical significance tests to detect subspace clusters. P3C defines significant regions as regions containing significantly more points than expected under a uniform-distribution assumption over a particular subspace. P3C first detects significant regions in individual dimensions by starting with intervals and merging adjacent significant regions together to form cluster cores. It then searches for higher dimensional regions using a bottom-up search strategy. Finally, P3C produces a matrix that records the probability of each point belonging to each cluster. P3C reports the results of the final clustering using an EM-like clustering procedure [14]. The advantage is that P3C can be used for both numerical and categorical data. Like other density-based algorithms,

P3C can detect arbitrarily shaped clusters and is robust to noise as well. The main disadvantage of P3C is that its performance depends on the detected one-dimensional clusters. If clusters have low density in their one-dimensional projections, there will not be enough statistical evidence for further aggregation.

Although the grid-based algorithms provide efficient clustering models, they cannot be applied to high-dimensional datasets, as the number of cells in the grid grows exponentially with the dimensionality of data. The cluster shape detected by grid-based clustering is a polygon (which can be an arbitrary shape for a high grid granularity) in the corresponding subspace where the space is divided in cells by a grid. Their accuracy and efficiency depend on the granularity and the positioning of the grid [2].

In addition, SUBCLU [15] and FIRES [16] are similar to CLIQUE, but apply different methods for clustering points. SUBCLU applies DBSCAN rather than grid-based clustering on each subspace candidate and outputs all detected clusters when searching for higher dimensional subspaces progressively. FIRES is a general framework for efficient subspace clustering. It runs a clustering algorithm on individual attributes separately in order to find "base clusters" and then utilises an approximation method for merging base clusters iteratively in order to find maximal attribute combinations in subspaces with higher dimensionality. After that, some methods can be used to refine the final clustering results. Its performance highly depends on the pruning and refinement steps [16].

Generally, clustering algorithms based on bottom-up searches normally result in overlapping/redundant clusters, such that a point can belong to many clusters simultaneously, making the clusters difficult to interpret. SUBCLU and the grid-based algorithms use a global density threshold, which leads to a bias against higher dimensional subspaces, i.e., a tighter threshold can filter noise well in low-dimensional subspaces but will lose clusters in higher dimensional subspaces. P3C can avoid this bias by using a threshold based on a statistical significance test, which can be treated as an adaptive density threshold. However, these bottom-up algorithms will take an extremely long time to discover subspace clusters if those clusters are high-dimensional, i.e., if the subspace they occupy spans many or most of the original attributes.

2.1.2. Top-down Methods

In order to find the best non-overlapping partitioning of the data into subspaces, most top-down methods aim to group points such that each point belongs to only one cluster and each cluster is assigned to a specific subspace [3]. This search method determines the subspace of a cluster starting from the full-dimensional space and then removes the irrelevant attributes iteratively. These methods usually evaluate each attribute for each point or cluster based on the "locality assumption" [6], i.e., the subspace of each cluster can be learnt from the local neighbourhood of cluster members. Note that

in some papers, subspace clustering with a top-down method is also called "projected clustering" [2].

In order to find non-overlapping data partitionings in subspaces, most top-down based subspace clustering algorithms aim to group points such that each point belongs to one cluster only; and each cluster is assigned to a specific subspace [3]. A top-down algorithm begins the search from the full-dimensional space and then removes the irrelevant attributes iteratively to find the target subspace.

PROCLUS [3] is a k-medoids type [17] clustering algorithm. It first randomly selects k potential cluster centres from the data sample and then iteratively refines them. In each loop of the iterative process, for each medoid, a set of dimensions is chosen by minimising the standard deviation of the distances of all points in the neighbourhood of the medoids to the corresponding medoid along every dimension. Points are then regrouped to their closest medoid based on the identified subspace of each medoid. In the final step of each loop, medoids are updated based on the points currently assigned to them. The iterative process continues until the clustering quality cannot be improved. The points which are too far away from their closest medoids are identified as noise. The output is k groups of non-overlapping points and a set of noise points. Each cluster has a globular shape in its identified subspace. The performance of PROCLUS is not stable since the initial medoids are randomly selected and are sensitive to irrelevant attributes in the full-dimensional space.

PreDeCon [18] is a top-down clustering algorithm which relies on DBSCAN [19]. For each point, this algorithm calculates a specialised subspace distance based on the concept of subspace preference, using it for density estimation and linking points. An attribute is relevant to the "subspace preference" of a point if the variance of data in the point's ϵ-neighbourhood is considered smaller than a threshold. It then links neighbouring points based on their density distribution on the weighted attributes. The downside of this approach is that there are three parameters that need to be manually set for density estimation [18]. Since it does not identify the subspaces where clusters exist but only weights attributes for each cluster, it is referred to as a "soft" subspace clustering algorithm (while others are called "hard" subspace clustering algorithms).

DOC [20] mixes the bottom-up approach for cluster detection and the top-down approach for iterative improvement in order to reduce redundancy. It uses a fixed density threshold with fixed side-length hypercubes to identify clusters.

2.2. Non-systematic Subspace Search

There are some subspace clustering algorithms which do not rely on a systematic search strategy but pursue heuristic approaches. For example, STATPC [21] formulates the search for statistically significant regions as subspace clusters with a grid-based structure. It defines the search as an optimisation problem and proposes a heuristic solution to avoid an exhaustive search.

In addition, there is rich literature on soft subspace clustering derived from k-means type [22] clustering. Different from selecting subspaces for each cluster, this kind of algorithms learn weight vectors over attributes with different objective functions and optimisation methods, and then incorporate these vectors for distance calculation for the k-means algorithm.

Generally, soft subspace clustering algorithms can be classified into three categories [23], i.e., conventional (all clusters share the same subspace, e.g., C-k-means [24]), independent (each cluster has an independent subspace, e.g., LAC [25], EWKM [26] and FSC [27]) and extended (enhanced performance for specific purposes, e.g., ESSC [28] and FG-k-means [29]). They also can be divided into soft subspace fuzzy clustering (e.g., FSC [27]) and soft subspace hard clustering (e.g., the five algorithms, apart from FSC, mentioned before), depending on whether each data object belongs to only one cluster or multiple clusters with different degrees [30, 31].

LAC, for example, starts with k centroids and k sets of d random weights, and approximates a set of k Gaussians by adapting the weights [25]. Soft subspace clustering algorithms can be seen as performing some type of normalisation of attributes for each cluster. In addition, k-means–based soft subspace clustering usually requires one parameter k as the number of clusters while other parameters can be set to default values [29].

EWKM is an entropy weighting subspace clustering algorithm which simultaneously maximises the negative weight entropy and minimises the within-cluster dispersion [26]. It gives different weights to attributes by utilising the weight entropy in the objective function. Therefore, clusters may have different selected attributes based on their weights.

Both ESSC and FG-k-means are based on EWKM method for specific purposes. ESSC integrates both within-cluster compactness and between-cluster separation information [28]. FG-k-means introduces multi-feature weighting for co-learning and multi-view learning [29]. They have better performance than EWKM, but FG-k-means and ESSC introduce 1 and 2 more parameters, respectively.

It is worth mentioning that there are many algorithms related with bi-clustering, co-clustering and pattern-based clustering [1]. They find clusters based on the pattern in the data matrix, and to cluster both objects and attributes simultaneously. They are usually aimed for genes and microarray datasets, and can only detect special types of subspace clusters [2]. They are not the focus of this paper.

2.3. Summary

Figure 2 compares properties of different subspace clustering algorithms. All existing subspace clustering tightly coupled their subspace selection process and clustering process; clustering definition cannot be changed easily. Note that the weights search in soft subspace clustering is also tightly coupled with the clustering process based on some optimisation functions. k-medoids and k-means type clustering algorithms usually cannot detect non-globular shaped clusters [2]. Soft subspace clustering,

Figure 2: Properties of subspace clustering algorithms and the proposed *CSSub* framework.

Algorithm	Subspace search strategy	Cluster subspace assignment	Cluster type	Arbitrarily shaped clusters	Non-overlapping clusters	Adaptive density threshold	Handling noise	Easy parameter setting	Clustering without original attributes
CLIQUE	Bottom-up	Hard	Grid	✓			✓		
MAFIA	Bottom-up	Hard	Grid	✓			✓		
ENCLUS	Bottom-up	Hard	Grid	✓			✓		
SSCS	Bottom-up	Hard	Density	✓	✓		✓		
P3C	Bottom-up	Hard	Grid	✓	✓	✓	✓	✓	
FIRES	Bottom-up	Hard	Flexible	✓		✓	✓		
SUBCLU	Bottom-up	Hard	Density	✓			✓		
PROCLUS	Top-down	Hard	k-medoids		✓		✓	✓	
PreDeCon	Top-down	Soft	Density	✓	✓		✓		
DOC	Hybrid	Hard	Grid	✓	✓				
STATPC	Heuristic	Hard	Grid	✓		✓	✓		
C-k-means	Optimisation	Soft	k-means		✓			✓	
FSC	Optimisation	Soft	k-means		✓			✓	
LAC	Optimisation	Soft	k-means		✓			✓	
EWKM	Optimisation	Soft	k-means		✓			✓	
ESSC	Optimisation	Soft	k-means		✓			✓	
FG-k-means	Optimisation	Soft	k-means		✓			✓	
CSSub	Non-search	Hard	Flexible	✓	✓	✓	✓	✓	✓

especially those based on k-means, cannot handle noise. Most density-based algorithms have more than 2 critical parameters for defining clusters, which are very difficult to set in practice.

This paper focuses on hard subspace and non-overlapping clustering. It is important to mention that the *CSSub* framework does not rely on any subspace search strategies or optimisation. Since *CSSub* clusters points based on the number of subspaces they shared, original attributes are not used for the similarity calculation in the clustering stage. In contrast, all the other algorithms rely on the original attributes.

3. Key weaknesses of existing bottom-up subspace clustering algorithms

The majority of density-based subspace clustering algorithms rely on a bottom-up search strategy and use anti-monotonicity of the density as a means to reduce the search space. This approach has two weaknesses:

1. The approach tightly couples the candidate subspace selection process with the clustering process,

i.e., it must run a clustering method for each subspace; and there are an exponentially large number of subspaces. As a result, many redundant subspace clusters are produced during the search process. This is a typical problem of using bottom-up search to find subspaces; and it has its origin in finding frequent item-sets [2]. While there are techniques to reduce the redundancy after the candidate subspaces have been generated, the approach is inefficient and ineffective in producing non-redundant subspace clusters.

2. The requirement to use a single (fixed) density threshold in order to make use of the anti-monotonicity property to prune the search space. Because density is a dimensionality biased measure [32], any global threshold selected will bias the search for subspace clusters toward low dimensional subspaces. While it is possible to use an adaptive density threshold or adaptive neighbourhood radius to detect more core points in higher dimensional subspaces [33], these methods cannot be directly deployed on density-based clustering algorithms. Incorporating these adaptive methods often involve ad hoc adjustments and with additional parameters.

To overcome the above weaknesses, we propose to use a different approach which does not rely on search and needs to run the clustering method only once to produce non-redundant subspaces.

4. Clustering by Shared Subspaces

In this section, we present an effective subspace clustering framework in order to overcome the weaknesses of existing bottom-up subspace clustering algorithms.

4.1. Definitions

Let D be a dataset of n points of d attributes, represented in a $n \times d$ matrix. A subspace cluster C_i is a submatrix of D with $|s_i| < d$ attributes, and $|C_i| \leq n$. We denote $S = \{s_1, s_2, ..., s_m\}$ as the set of subspaces to be explored. Let $\pi_s(x)$ denote the point $x \in D$ projected on a subspace $s \in S$. Let the set of k subspace clusters in D be $C = \{C_1, C_2, ..., C_k\}$.

Definition 1. *Point x is an α-**Core** point in subspace s if x has a high score value based on a scoring function $\alpha(\cdot)$ in s, i.e., $(\alpha_s(x) > \tau_s) \leftrightarrow \alpha\text{-}Core_s(x)$. Subspace s is thus a candidate subspace (core-point subspace) for x in which a cluster containing x may exist.*

Note that τ_s depends on subspace s; and it is a not a global threshold for all subspaces. Noise points are non-α-Core points in all subspaces considered.

Definition 2. *Similarity by **shared subspaces**: the similarity between x_i and x_j is defined as the number of shared subspaces in which both x_i and x_j are α-Core.*

For example, if density is used as the score $\alpha(\cdot)$, then the similarity between x_i and x_j is defined as the number of subspaces in which they are both in dense locations.

Let $A(x) = \{s \in S \mid \alpha\text{-}Core_s(x)\ \}$ be the set of subspaces where point x is an α-Core point. As each point is α-Core in a set of subspaces which is different from that of another point, the similarity is formulated through normalisation as follows:

$$sim(x_i, x_j) = \frac{|A(x_i) \cap A(x_j)|}{|\ A(x_i) \cup A(x_j)|}$$

Note that the above formulation is the Jaccard similarity [34]; it can be interpreted as the higher the probability that two α-Core points are in shared subspaces, the more similar the two points are.

Definition 3. *Cluster C exists in a set of shared subspaces S if $\forall_{i \neq j,\ x_i, x_j \in C}\ sim(x_i, x_j) \geq \beta_C$, where β_C is a threshold for cluster C.*

Note that β_C varies for each cluster and it can be determined automatically if a certain clustering algorithm is employed, e.g., a partitioning-based algorithm k-medoids. Hereafter we denote α-Core points as "core points".

4.2. Clustering Using Shared Subspaces

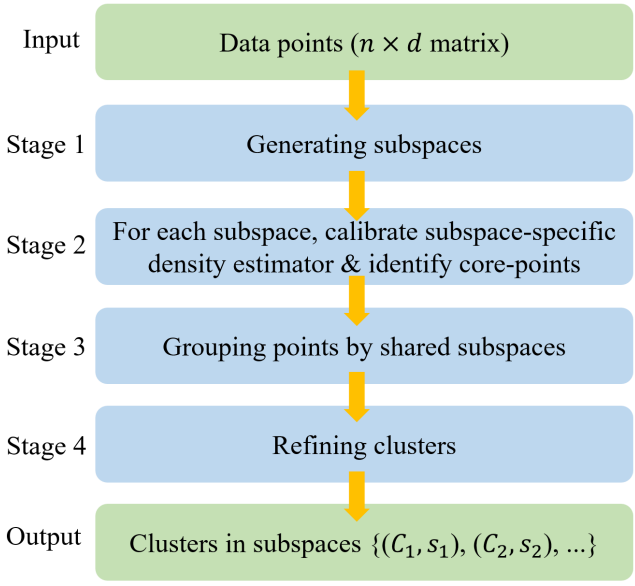


Figure 3: A conceptual overview of the *CSSub* framework for subspace clustering.

A conceptual overview of the *CSSub* framework is shown in Figure 3. There are four stages. In Stage 1, we first generate an initial set of subspaces for assessing. Then each point is examined whether it is a core point in each of the subspaces in Stage 2. Candidate subspaces are selected if they have

core points. Non-core points in all subspaces are identified as noise. In Stage 3, by using a similarity measure based on shared subspaces, such as the Jaccard similarity, an existing clustering algorithm can be employed to perform clustering on the points defined using the candidate subspaces identified in Stage 2. In Stage 4, some methods are applied to refine these clusters, e.g., finding the best subspace for each cluster to produce non-overlapping clusters.

4.2.1. Scoring Function

CSSub can produce different types of clusters depending on the scoring function employed. We first use a density scoring function to demonstrate the ability of *CSSub* to detect arbitrarily shaped clusters. We then show that a different score can be easily used in the framework in Section 6.4.

To use a density function to identify core points, a density estimator is required. We employ the ϵ-neighbourhood density estimator [19] as follows:

$$\widehat{pdf}^{\epsilon}_{s}(x) = \frac{1}{nV^{\epsilon}_{d}} \sum_{j} \mathbf{1}[\|\pi_s(x) - \pi_s(x_j)\| \leqslant \epsilon] \tag{1}$$

where $\mathbf{1}[\cdot]$ denotes the indicator function and $\pi_s(x)$ is point $x \in D$ projected on subspace $s \in S$, and $V^{\epsilon}_{d} \propto \epsilon^d$ is the volume of a d-dimensional ball of radius ϵ. As the volume is constant for all points, it is usually omitted in actual computations.

Since data become sparse as the dimensionality of subspace increases, a global ϵ is not appropriate, i.e., using a relatively very small or very large ϵ may get a uniform density distribution for each point in a subspace. In order to maximally differentiate points in terms of their estimated density values in a subspace s, we use an adaptive ϵ_s setting to obtain the maximum variance of the normalised densities such that: $\epsilon_s = \arg\max_{\epsilon \in R} \sum_j (p^{\epsilon}_{s}(x_j) - \frac{1}{n}\sum_k p^{\epsilon}_{s}(x_k))^2$, where $p^{\epsilon}_{s}(x) = \widehat{pdf}^{\epsilon}_{s}(x) / \sum_y (\widehat{pdf}^{\epsilon}_{s}(y))$ is the normalised density of x in subspace s given ϵ. In our implementation, a fix number g of possible ϵ values is examined to obtain the best value ϵ_s for each subspace.

Let $p^{\epsilon_s}_{s}(x)$ be the density score of point x in subspace $s \in S$. We use $p^{\epsilon_s}_{s}(\cdot)$ as the subspace scoring function in Stage 2. Subspace s is a selected candidate subspace for x if $(p^{\epsilon_s}_{s}(x) > \tau_s)$. Note that $p^{\epsilon_s}_{s}(\cdot)$ is used in place of $\alpha_s(\cdot)$ in Definition 1 to determine core points in subspace s. Here we propose a heuristic method that set τ_s to the average score in each subspace as the default parameter, i.e., $\tau_s = \frac{1}{n}\sum_{x \in D} \alpha_s(x) = \frac{1}{n}\sum_{x \in D} p^{\epsilon_s}_{s}(x)$. This means that every core point has a density larger than the average density in the subspace. Thus, parameters ϵ_s and τ_s in this density-based scoring function are set automatically.

4.2.2. Grouping Algorithm

A traditional clustering algorithm can be used to group points based on their Jaccard similarities. We focus on discovering non-overlapping clusters C such that $\forall_{i \neq j, C_i, C_j \in C}\ C_i \cap C_j = \emptyset$. In this paper, we use k-medoids algorithm [17] for finding k optimal non-overlapping partitions.

Algorithm 1 *Clustering_by_Shared_Subspaces*(D, k)

Input: D: input data ($n \times d$ matrix); k: number of clusters.

Output: $\{C_i, s_i\}_{i=1}^{k}$: subspace clusters; *noise*: noise points.

/* *Stage 1:* */

$\mathsf{d} \leftarrow \max dim \; s.t. \; \sum_{i=1}^{dim} \binom{d}{i} < n$;

$S \leftarrow generate\_subspaces(D, \mathsf{d})$;

/* *Stage 2:* */

for $s \in S$ **do**

  $\epsilon_s \leftarrow \arg\max_{\epsilon \in R} \sum_j (p_s^{\epsilon}(x_j) - \frac{1}{n} \sum_k p_s^{\epsilon}(x_k))^2$;

  $core_{s,j} \leftarrow \mathbf{1}[\; p_s^{\epsilon_s}(x_j) > \frac{1}{n} \sum_k p_s^{\epsilon_s}(x_k) \;] \quad \forall_{j=1}^{n}$;

end for

$noise \leftarrow \{x_j \in D \mid \forall_{s \in S} \;\; core_{s,j} = 0\}$;

$core \leftarrow \forall_{s,j}$ remove $cores_{s,j}$ if $x_j \in noise$;

/* *Stage 3:* */

$\{C_i\}_{i=1}^{k} \leftarrow k\_medoids(core, k)$;

/* *Stage 4:* */

for $i \in \{1, \ldots, k\}$ **do**

  $shared_i = \{s \in S \mid \exists_{x_j \in C_i} \;\; core_{s,j} = 1\}$;

  $s_i \leftarrow \arg\max_{s \in shared_i} \sum_{x_j \in C_i} core_{s,j}$;

end for

return $\{C_i, s_i\}_{i=1}^{k}$; *noise*;

k-medoids is more resilient to noise and outliers than k-means. Instead of using the mean value to represent a cluster, k-medoids [17] chooses one data points $c_i \in C_i$ as the centre (medoid) for a cluster. A medoid is a point with the minimal average distance to all the points in the cluster. Thus, the best medoid for each cluster is found by minimising the sum of the dissimilarities between each point in the cluster and a candidate medoid.

5. Algorithms in *CSSub*

In this section, we provide the algorithms for the proposed *CSSub* framework shown in Figure 3. The stages shown in Algorithm 1 correspond to the stages in Figure 3.

The first stage in Algorithm 1 generates the set of subspaces by enumeration to the largest subspace dimensionality $\mathsf{d} \geqslant 1$ such that $|S| = \sum_{i=1}^{\mathsf{d}} \binom{d}{i} = m < n$. Here we set the number of candidate subspaces less than the data size is to keep the quadratic time complexity. Different ways to generate the initial set of subspaces may be used instead in the framework, discussed in Section 7.1.

Algorithm 2 *k_medoids*($core$, k)

Input: *core*: core-point subspace matrix; k: number of desired clusters.

Output: C: a set of clusters.

1: Randomly select k data points from *core* as the medoids;

2: Each data point from *core* is assigned to the closest medoid to form clusters $C_i, i = 1, \ldots, k$ based on the Jaccard similarity;

3: Revise the medoid in each cluster which yields the maximum sum of the similarity between each point in the cluster and the medoid, i.e., for each cluster i, find medoid $M_i = \arg\max_{y \in C_i} \sum_{x \in C_i} sim(x, y)$;

4: Repeat steps 2 and 3 until there is no change in point reassignments;

5: **return** $C = \{C_1, \ldots, C_k\}$;

In Stage 2 of Algorithm 1, the output *core* is a binary matrix, which indicates whether a point is a core point in a subspace. It also outputs the noise points which are not core points in any subspaces assessed; they are excluded for clustering.

In Stage 3 of Algorithm 1, we use a k-medoids algorithm based on the PAM (Partitioning Around Medoids) algorithm [17] to cluster the data on the matrix *core* using the Jaccard similarity. The k-medoids algorithm is shown in Algorithm 2.

For each detected cluster $C_i \subset C$ at the end of Stage 3 in Algorithm 1, there exists a set of shared subspaces $shared_i = \{s \in S \mid \alpha\text{-}Core_s(x) \wedge (x \in C_i)\}$.

In Stage 4 of Algorithm 1, we refine the clustering result of k-medoids in order to find the best subspace for each cluster. The final subspace for C_i is the one in $shared_i$ which covers the largest number of points in C_i, i.e., $s_i = \arg\max_{s \in shared_i} |\{x \in C_i \mid \alpha\text{-}Core_s(x)\}|$.

The final clustering outputs are the k clusters: $\{(C_1, s_1), (C_2, s_2), ..., (C_k, s_k)\}$; and noise points identified in Stage 2.

It is interesting to note that in using k-medoids to group points by shared subspaces, the algorithm implicitly sets β_{C_i} in Definition 3 automatically for each cluster C_i having medoid M_i as follows: $\beta_{C_i} = \min_{x \in C_i} sim(x, M_i)$.

CSSub requires only one parameter k to be set manually, and has the total time complexity of $O(gn^2d^{\mathsf{d}})$.

6. Empirical Evaluation

This section presents experiments designed to evaluate the performance of *CSSub*. We selected seven state-of-the-art non-overlapping subspace clustering algorithms for comparison: a top-down subspace clustering algorithm PROCLUS [3], a bottom-up subspace clustering algorithm P3C [13] and

five soft subspace clustering algorithms LAC [25], EWKM [26], FSC [27] ESSC [28] and FG-k-means [29]. In addition, we include a full-space clustering algorithms k-medoids [17] in order to judge the effectiveness of subspace clustering for real world tasks. Because it is difficult to assess the clustering performance for algorithms which produce many redundant clusters, this type of subspace clustering algorithms is omitted in the comparison.

We used 3 synthetic datasets (2T, S1500 and D50) and 12 real-world datasets[4] to ascertain the ability of different subspace clustering algorithms in handling different kinds of datasets. All datasets are normalised using the *min-max* normalisation. The properties of these datasets are presented in Table 1.

Table 1: The properties of 15 datasets. The first 3 datasets are synthetic datasets and the rest are real-world datasets.

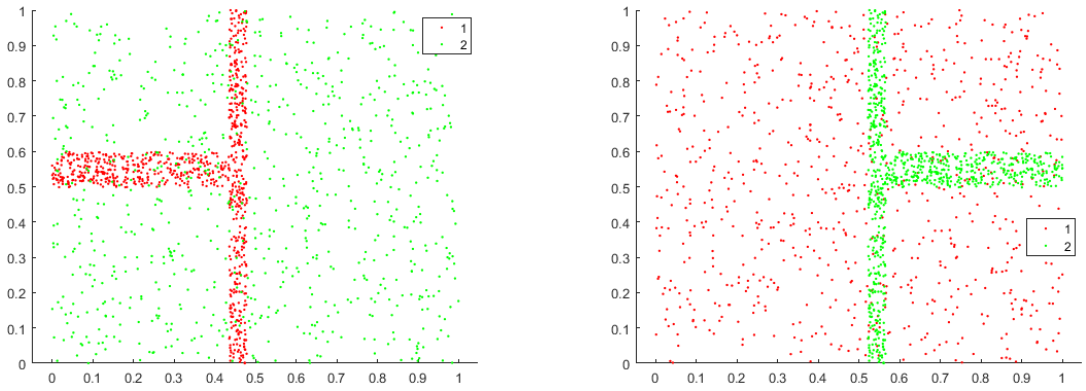
Datasets	#Points	#Dimensions	#Clusters
2T	1600	4	2
S1500	1595	20	12
D50	1596	50	13
Glass	214	9	6
Liver	345	6	2
ILPD	579	9	2
Musk	476	166	2
Sonar	208	60	2
Ionosphere	351	33	2
Spam	4601	57	2
Iris	150	4	3
Wine	178	13	3
Dermatology	358	34	6
WDBC	596	30	2
Breast	699	9	2

The 2T synthetic dataset contains two T-shaped clusters embedded in a four-dimensional space, while the S1500 and D50 synthetic datasets have 12 and 13 Gaussian clusters, respectively, embedded in different subspaces. Every irrelevant attribute in these synthetic datasets is uniformly distributed between 0 and 1. Figure 4 and Figure 5 show the data distributions of the 2T and S1500 datasets in two different subspaces, respectively. The D50 dataset has 50 attributes and more irrelevant attributes than the S1500 dataset.

PROCLUS, FG-k-means, LAC and P3C were implemented in Weka, and all other algorithms were implemented in Matlab [5]. The experiments were run on a machine with eight cores (Intel Core i7-7820X 3.60GHz) and 32GB memory. All datasets were normalised using the *min-max* normalisation

[4]Synthetic datasets S1500 and D50 are from [8]. All real-world datasets are from UCI Machine Learning Repository [35].

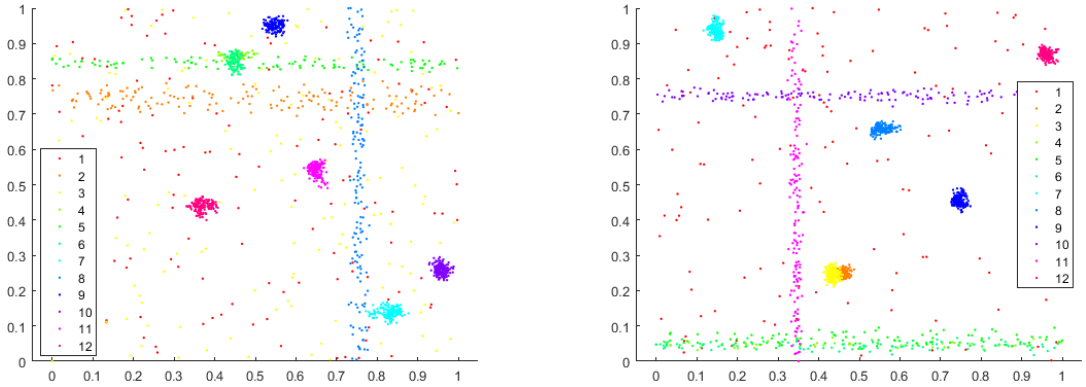
[5]The source codes of PROCLUS and P3C are from the paper [8]. FG-k-means and LAC source codes are from the paper [29]. Other soft subspace clustering algorithms were from the paper [23].



(a) Plotted in attribute 1 & 2.

(b) Plotted in attribute 3 & 4.

Figure 4: Distributions of the 2T dataset in two subspaces.



(a) Plotted in attribute 1 & 2.

(b) Plotted in attribute 3 & 4.

Figure 5: Distributions of the S1500 dataset in two subspaces.

to yield each attribute to be in [0,1] before the experiments began.

The parameter settings used for all clustering algorithms under comparison are shown in Table 2. They are reasonable settings as used in the literature (e.g., [8, 23, 25, 29]). For all k-means and k-medoids type clustering algorithms, we ran these algorithms over 10 trials using different random seeds for initial k centres in order to obtain their best performance.

We evaluate the clustering performance in terms of Macro F-measure, since the ground truth subspaces for each cluster are not available for real-world datasets.[6] Given a clustering result, we

[6]External evaluation measures such as Purity and NMI [2] only take into account the points assigned to clusters. A clustering, which assigns the majority of the points to noise, can produce a high score in terms of purity or NMI. F-measure is more suitable than purity and NMI in assessing clustering performance of clustering algorithms which identify noise, as in the case for subspace clustering algorithms.

Table 2: Parameters and assigned value or search range. Note that $|C|$ is the true number of clusters.

Algorithm	Parameters and assigned value or search range
$CSSub$	$k = \|C\|$
$CSSub_I$	$k = \|C\|$; $\psi = 256$; $t = 100$
PROCLUS	$k = \|C\|$; $avgDim = 0.5 \times d$
P3C	$\alpha = 0.001$; $poisson \in \{5, 10, ..., 100\}$
EWKM	$k = \|C\|$; $\gamma = 10$
LAC	$k = \|C\|$; $h = 10$
ESSC	$k = \|C\|$; $\gamma = 10$; $\eta = 0.5$; $m = s/(s-2)$, where $s = min(n, d)$
FSC	$k = \|C\|$; $\tau = 10$; $\epsilon_0 = 10^{-5}$
FG-k-means	$k = \|C\|$; $\eta = 10$; $\lambda = 10$
k-medoids	$k = \|C\|$

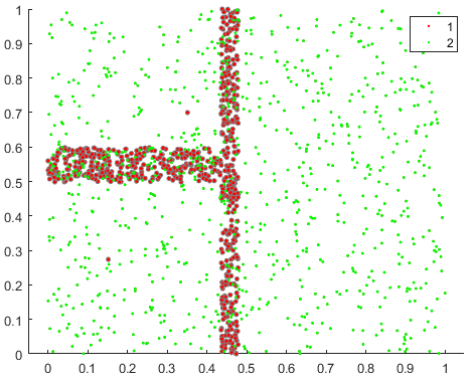
calculate the precision score P_i and the recall score R_i for each cluster C_i based on the confusion matrix with the Hungarian method [36] for finding the maximum weight matching in a bipartite graph. The F-measure of C_i is the harmonic mean of P_i and R_i. The overall F-measure is the average over all clusters: F-measure$= \frac{1}{k}\sum_{i=1}^{k}\frac{2P_iR_i}{P_i+R_i}$.

6.1. Illustrative Clustering Performance Using a Synthetic Dataset

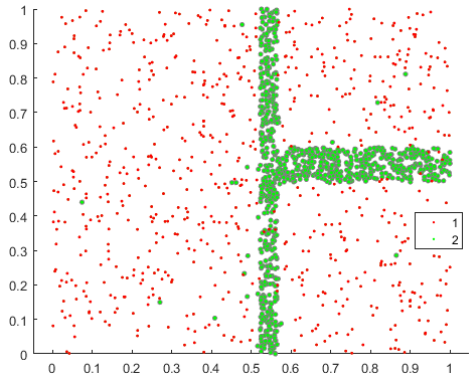
In this section, the 2T synthetic dataset is used to examine whether a subspace clustering algorithm can detect arbitrarily shaped clusters in subspaces of different dimensions.

Figure 6 to Figure 8 show clusters detected by the three hard subspace algorithms on the 2T dataset. Although all algorithms correctly identified the subspace for each cluster on 2T, the number of correctly assigned points differs greatly among these algorithms. *CSSub* incorrectly assigned a few points only (see Figure 6). Figure 7 shows that the clusters detected by PROCLUS have globular shapes—this is the known weakness of k-medoids and k-means type algorithms when clustering algorithms are performed on the given feature space—they are unable to detect arbitrarily shaped clusters. Because P3C utilises a bottom-up based subspace search strategy, it may incorrectly group neighbouring points which only have high densities in low dimensional projections, as shown in Figure 8. The overall result is reflected in F-measure; *CSSub* has 0.94, which is much higher than those obtained by PROCLUS (0.79) and P3C (0.76).

The relative results for the other two synthetic datasets are similar, i.e., *CSSub* has better F-measure than all other algorithms. These results are summarised in the first row of Table 3. The superior clustering performances of *CSSub* in detecting clusters in subspaces are mainly due to the

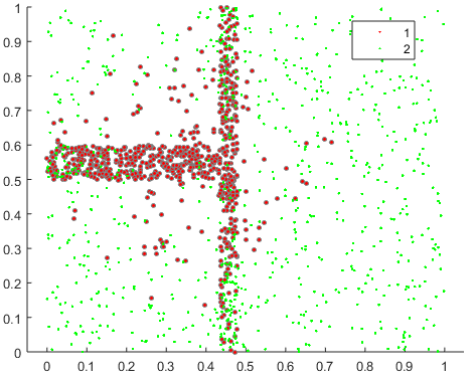


(a) Plotted on attributes 1 & 2.

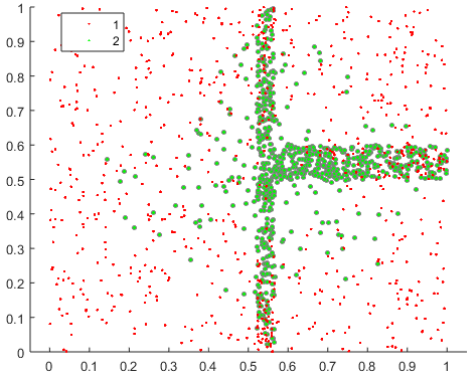


(b) Plotted on attributes 3 & 4.

Figure 6: Clusters labelled by *CSSub* on the 2T dataset.

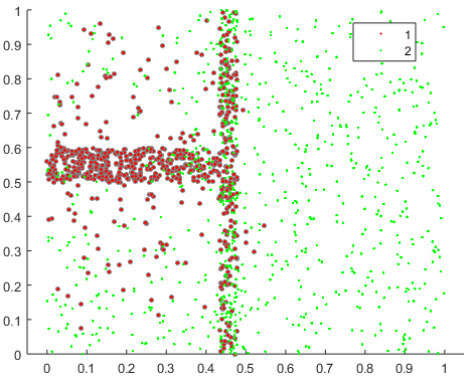


(a) Plotted on attributes 1 & 2.

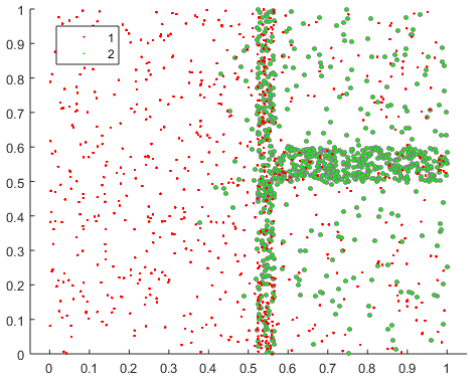


(b) Plotted on attributes 3 & 4.

Figure 7: Clusters labelled by PROCLUS on the 2T dataset.



(a) Plotted on attributes 1 & 2.



(b) Plotted on attributes 3 & 4.

Figure 8: Clusters labelled by P3C on the 2T dataset.

use of clustering by shared subspaces—the distinctive feature of *CSSub* in comparison with other subspace clustering algorithms. Despite the fact that the same k-medoids algorithm is used in *CSSub*, the use of clustering by shared subspaces has enabled it to detect arbitrarily shaped clusters; while it is impossible when k-medoids clustering is performed on attribute values of the given dataset.

6.2. Clustering Performance on All Datasets

In this section, we present the clustering results on all synthetic and real-world datasets. Table 3 shows the best F-measure on 15 datasets.

As we do not know whether the real-world datasets have clusters in subspaces, we perform a sanity check: sort the results based on the full-space clustering k-medoids algorithm—the datasets in which it has high F-measure are unlikely to have clusters in subspaces. The real world datasets in Table 3 are split into two subsets: one above F-measure = 0.90 and one below.

Table 3: Best F-measure of different clustering algorithms on 15 datasets; the result of real-world datasets are sorted by F-measure of k-medoids. The best two performers on each dataset are boldface. The last 5 datasets are greyed because full-space clustering k-medoids have performed well, indicating that they have no clusters in subspaces. $CSSub_I$ is *CSSub* with a linear time scoring function, discussed in Section 6.4. The first 3 datasets are synthetic datasets and the rest are real-world datasets. The average F-measure is over the first 10 datasets.

Dataset	**Hard subspace clustering**				**Soft subspace clustering**					**Full**
Name	*CSSub*	$CSSub_I$	PROCLUS	P3C	EWKM	LAC	ESSC	FSC	FG-k-means	k-medoids
2T	**0.94**	**0.92**	0.79	0.76	0.73	0.64	0.78	0.68	0.78	0.69
S1500	**0.96**	**0.89**	0.75	0.66	0.65	0.69	0.75	0.77	0.70	0.68
D50	**0.89**	**0.84**	0.67	0.59	0.65	0.78	0.72	0.81	0.65	0.62
Glass	**0.46**	0.43	0.40	0.28	0.40	0.29	**0.44**	0.33	0.37	0.33
Liver	0.49	0.47	**0.54**	0.36	0.43	0.46	0.44	0.44	0.43	**0.50**
ILPD	**0.59**	0.44	0.56	**0.58**	0.56	0.56	0.57	0.57	0.56	0.51
Musk	0.49	0.53	0.51	0.36	0.55	0.53	0.55	**0.56**	**0.56**	0.54
Sonar	0.55	0.57	**0.60**	0.46	0.49	0.52	0.56	**0.61**	0.51	0.55
Ionosphere	**0.78**	**0.75**	0.73	0.28	0.70	0.70	0.70	0.68	0.57	0.70
Spam	0.76	0.75	0.69	0.38	0.63	0.37	**0.81**	**0.85**	0.42	0.73
Iris	0.40	0.35	0.86	0.56	**0.90**	0.88	0.90	0.96	0.89	**0.90**
Wine	0.71	0.57	0.88	**0.93**	0.95	0.95	0.95	0.94	**0.92**	0.91
Dermatology	0.70	0.73	**0.86**	0.56	0.67	0.52	0.96	0.55	0.46	**0.92**
WDBC	0.87	0.73	0.84	0.59	0.92	0.92	0.92	**0.94**	0.92	**0.93**
Breast	0.80	0.76	0.90	0.90	0.95	0.95	**0.95**	0.95	**0.95**	0.95
Average	0.69	0.66	0.62	0.47	0.58	0.55	0.63	0.63	0.55	0.59

It is indeed the case that few of the subspace clustering algorithms can perform better than k-medoids full-space clustering in datasets having F-measure > 0.90. We can safely assume that the last 5 datasets in Table 3 have no clusters in subspaces. We thus focus our comparison hereafter on the first 10 datasets in which clusters may exist in subspaces to examine the capability of subspace clustering algorithms.

Note that most hard subspace clustering algorithms assume that clusters exist in different subspaces. If all clusters can be separated in full space only, then they may perform badly because some important attribute values may be lost in (dis)similarity calculation. We have observed this phenomenon on some real-world datasets.

On the 10 datasets, *CSSub* has the highest average F-measure of 0.69, as shown in Table 3. Both ESSC and FSC have the average F-measure of 0.63. The average F-measure of PROCLUS and P3C are 0.62 and 0.47, respectively. It is interesting to note that the average F-measure of k-medoids is 0.59, which is slightly higher than that of EWKM, LAC and FG-k-means.

We conduct the Friedman test with the post-hoc Nemenyi test [37] to examine whether the difference between any two subspace clustering algorithms is significant in terms of their average ranks. Figure 9 shows five subspace clustering algorithms' average ranks and critical difference. This test shows that *CSSub*, $CSSub_I$ and PROCLUS are significantly better than P3C. *CSSub* is significantly better than FG-k-means as well. ESSC is the best performer of soft subspace clustering algorithms, followed by FSC.

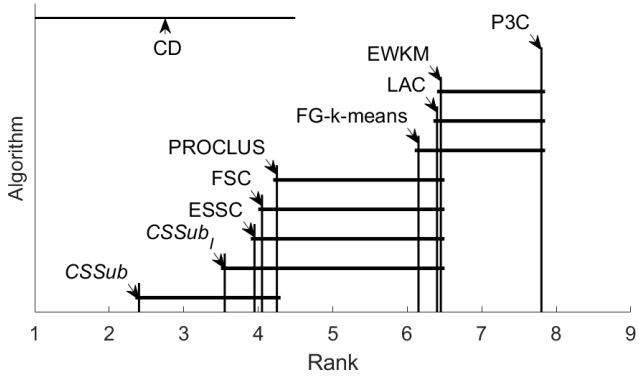


Figure 9: Critical difference (CD) diagram of the post-hoc Nemenyi test ($\alpha = 0.10$). The difference between two algorithms is significant if the gap between their ranks is larger than CD. There is a line between two algorithms if the rank gap between them is smaller than CD.

Generally, hard subspace clustering algorithms usually perform poorly or even cannot run in a reasonable time on high-dimensional datasets (thousands of dimensions) [7]. *CSSub* can perform well on low-dimensional to medium-dimensional datasets (tens of dimensions) with $n \gg d$. This is because it examines a large proportion of all possible subspaces to identify clusters. In addition, density estimation is more reliable in low-dimensional spaces, while it becomes difficult in high-dimensional

[7] Although many existing hard subspace clustering algorithms are claimed to be able to deal with "high-dimensional" datasets. This usually refers to tens of dimensions only [1, 8, 38]. In this paper, we refer these "high-dimensional" datasets as medium-dimensional datasets.

spaces since these spaces are mostly empty [39].

Different from hard subspace clustering algorithms, many soft subspace clustering algorithms are designed to cluster high-dimensional datasets since they usually have the linear time complexity in both data size and dimensionality [23]. The examples of experimental results on real-world high-dimensional datasets are given in the Appendix B.

6.3. Scalability

We test the scalability of the subspace clustering algorithms on a synthetic dataset with increasing data size. For a fair evaluation, we set their parameters to explore the same number of subspaces, i.e., the maximum search depth is equal to the number of dimensions of the dataset used. Note that since all the five soft subspace clustering algorithms have the linear time complexity to the data size, we only report the evaluation results of ESSC as the soft subspace clustering representative.

Table 4 presents the runtime of the scalability test. The runtime ratio of this test is shown in Figure 10. For the increasing data size test, *CSSub*, PROCLUS and P3C have the trend of quadratic time whereas ESSC has a linear time trend. *CSSub* has the worst runtime due to the searching of best values ϵ_s for density estimation in each subspace. However, we show in the next section that its runtime can be significantly improved by simply replacing the quadratic time density score with a linear time scoring function.

Table 4: Runtime (in seconds) of the scalability test wrt the number of points on a synthetic dataset with 4 attributes. *CSSub*, $CSSub_I$ and ESSC were tested in Matlab, while PROCLUS and P3C were tested in Weka.

Data size	*CSSub*	$CSSub_I$	PROCLUS	ESSC	P3C
500	0.62	0.59	0.01	0.01	0.03
1500	4.76	0.68	0.03	0.01	0.21
2500	12.90	0.84	0.06	0.02	0.48
3500	25.11	0.97	0.10	0.03	0.92
4500	41.48	1.08	0.15	0.03	1.51
5500	62.11	1.33	0.29	0.04	2.35

6.4. CSSub with a Linear Time Scoring Function

CSSub is flexible enough to employ various methods for defining clusters. It can use different scoring functions to create different types of clusters.

In addition, although the overall time complexity of *CSSub* is quadratic to data size, the total runtime of *CSSub* is dominated by the subspace assessment. The flexibility to use a different scoring function enables *CSSub* to easily convert from one with quadratic time complexity to one with linear time complexity by using a linear scoring function.

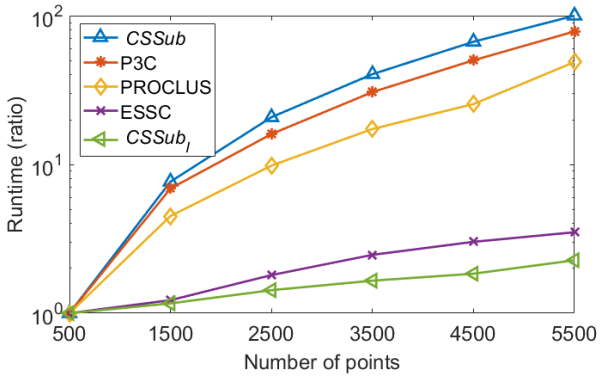


Figure 10: Runtime ratio of Scalability test.

Instead of using density score, $CSSub_I$ uses a linear scoring function based on isolation path length score[8]. This score was previously used to assess subspaces for outlying aspect mining [32].

An $iForest$ (consists of t $iTrees$) is generated for each subspace. Each $iTrees$ (isolation tree) is a binary tree and is built independently using a subset $\mathcal{D}$, sampled without replacement from $\pi_s(D)$, where $|\mathcal{D}| = \psi$. The idea is to isolate each point from the rest of the points in the sample. This is done in a random process. At each internal node of a tree, it generates a binary splitting by random selecting an attribute and its splitting point, and then splits the sample into the two children nodes. The process is repeated recursively until no further splits can be made.

The path length of a query x on an $iTree$ is the number of nodes x traverses from the root node to a leaf node.The final isolation path length score $L_s(x)$ of x in a subspace $s \in S$ is the average path length over t $iTrees$. We set ψ=256 and t=100 as the default parameter for building $iForest$, as suggested by Liu et al [40]. The details of the $iTree$ building process and path length score calculation are provided in the Appendix A.

The advantages of using isolation path length as scoring function are (i) it is dimensionality unbiased [32]; and (ii) it can be computed more efficiently than density estimation—the time complexity for scoring all points in a subspace is $O(nt \cdot log(\psi))$.

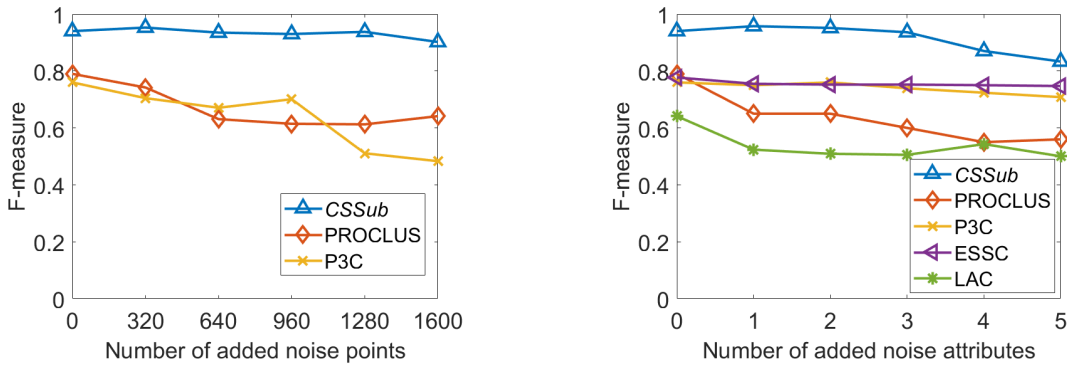
We found that $CSSub_I$ is a promising method which improves $CSSub$'s runtime significantly and exhibits linear time behaviour wrt data size, as shown in Figure 10. In addition, $CSSub_I$ has

[8]This score is derived from $iForest$ (isolation forest) [40], originally used for anomaly detection as the anomaly score. $iForest$ identifies anomalies as points (located in sparse regions) having short average path lengths, while normal points (located in dense regions) having long average path lengths. Because short path lengths indicate anomalies and long path lengths indicate normal points, a cluster can then be defined using neighbouring points which have path lengths longer than a certain threshold.

clustering accuracy with the average F-measure of 0.66 on the 10 datasets, which is only slightly worse than *CSSub* with F-measure of 0.69, as shown in Table 3.

6.5. Robustness Against Noise

CSSub is robust against noise because it uses an adaptive density threshold for core point identification in each subspace. The noise points were not selected by *CSSub* as core points in most subspaces. As a result, the noise points have little or no effect on its clustering result. In terms of noise attributes, *CSSub* can still perform well when only a small number of shared subspaces contain noise attributes, because the dissimilarity calculation is based on many shared subspaces. We examined this capability by adding noise points and noise attributes (separately) to the 2T dataset, and found that *CSSub* can tolerate the same level of noise as least as good as other subspace clustering algorithms, as shown in Figure 11.



(a) 2T with noise points

(b) 2T with noise attributes

Figure 11: Best F-measure on the 2T dataset with added noise points and noise attributes. A noise point or noise attribute has values uniformly distributed in [0,1] in each attribute. Soft subspace clustering are omitted in (a) as they are very sensitive to noise points, similar to k-means. FG-k-means, EWKM and FSC are omitted in (b) as they perform very similar to ESSC.

7. Discussion

CSSub is effective on low-to-medium dimensional datasets having subspace clusters with different irrelevant attributes. The experiment using the three artificial datasets, 2T, S1500 and D50, indicates that *CSSub* is particularly good at identifying subspace clusters of non-globular shape. The robustness analysis shows that it is tolerant to both noise points and noise attributes.

The *CSSub* framework provides a flexible platform for developing new subspace clustering algorithms which group points by shared subspaces. It allows different combinations of components from

either new or existing clustering algorithms. In the following, we discuss additional features of *CSSub*, potential improvements and research directions based on this framework.

7.1. Subspace generation methods

CSSub has the flexibility of selecting systematic or non-systematic ways to generate subspaces in Stage 1. In the experiments reported thus far, the set of subspaces was generated by exhaustive enumeration to the largest subspace dimensionality d (which is set automatically). Since this subspace generation process has the time complexity of $O(d^{\mathsf{d}})$, it is infeasible on datasets with large dimensionality.

An alternative is to use a beam search method [41] for subspace generation. We design a heuristic search procedure using the beam search with the beam width W and the stage level l, described as follows. In the first level, all 1-dimensional subspaces are assigned scores and then only the top W subspaces with the highest scores are kept for the next level. Here, the score for each subspace is the maximum variance of the normalised densities, obtained from the same adaptive ϵ-density estimator as discussed in Section 4. In the subsequent level $(l+1)$, we combine the W subspaces from level l in a pairwise fashion, producing new subspaces containing the union of the attributes present in the two constituent subspaces from the previous iteration. This process generates up to $\binom{W}{2}$ new subspaces. Each subspace is assigned scores and ranked; and only the top W subspaces with the highest scores are kept for the next level. This procedure continues until a user-defined maximum level H is reached. The W top ranked subspaces from each level $l = \{1, ..., H\}$ are then combined to form the set of candidate subspaces (of maximum size HW) for use in clustering. The time complexity of this beam search method is $O(d + HW^2)$. Note that the proposed use of the beam search method is to generate subspaces only in Stage 1, the other stages remain the same for *CSSub*.

Figure 12 shows the *CSSub* clustering results using the above beam search for subspace generation on the D50 dataset. We set the maximum level $H = 10$, and vary the beam width W from 5 to 50. It can be seen from Figure 12 that both F-measure and maximum subspace dimension increase as W increases to 30.

To compare the performance of *CSSub* using the beam search and *CSSub* using the exhaustive enumeration for subspace generation, we tested them on the D50 dataset with different numbers of added noise attributes, as shown in Figure 13. We set $W = 30$ and $H = 10$ for the beam search method, and manually set $\mathsf{d} = 2$ for the enumeration method. It is clear that both methods are tolerant to noise attributes. The key difference is that *CSSub* using the beam search is linear to data dimensionality; but the one using the enumeration is exponential to dimensionality, i.e., the runtime is increased by a factor of 4 when the dimensionality is doubled from 50 to 100 (having additional 50 irrelevant attributes).

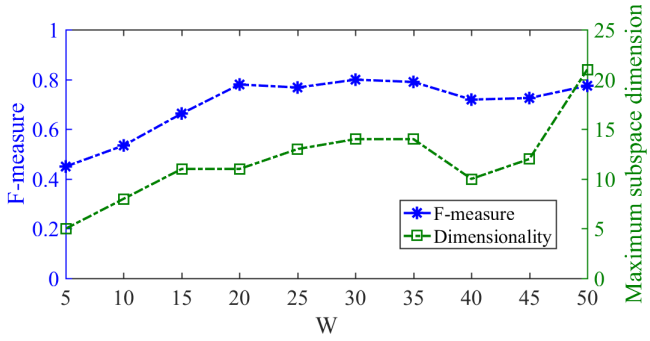
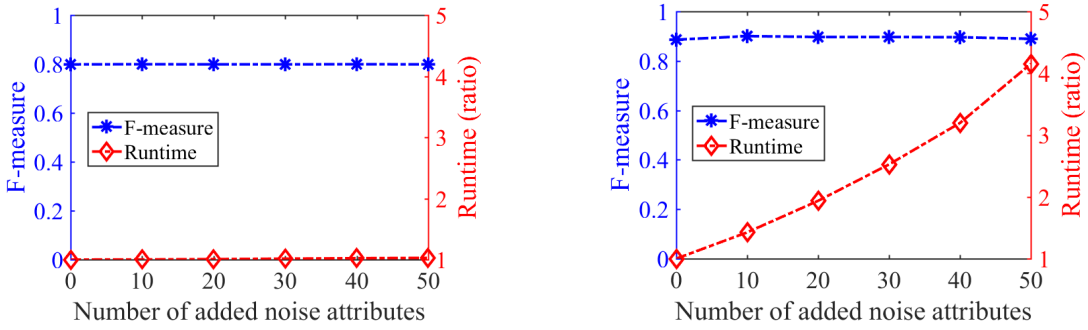


Figure 12: *CSSub* clustering results with a beam search for subspace generation on D50 dataset.



(a) *CSSub* with beam search

(b) *CSSub* with enumeration

Figure 13: F-measure and runtime on the D50 dataset with added noise attributes. Each noise attribute has values uniformly distributed in [0,1]. (a) *CSSub* using the beam search for subspace generation, (b) *CSSub* using the enumeration method for subspace generation.

The beam search method has the advantage of significantly shorter runtime, especially in high dimensionality. But, it has two parameters and may lead to poor clustering results because it may lose important low-dimensional subspaces at low levels of the search. Therefore, we only recommend this method on medium-dimensional datasets.

7.2. Other unique features of CSSub

There are two other unique features of *CSSub* which have not been mentioned thus far.

First, a recent work [32] has shown that it is inappropriate to use a density score function to assess subspaces in an algorithm which employs systematic search for subspaces because a density score is a dimensionality biased function which biases toward low dimensional spaces. Existing subspace clustering methods that employ systematic search should use a criterion other than density to assess subspaces, even though they use a density-based clustering algorithm to perform clustering. For example, ENCLUS employs an entropy-based method for subspace assessment while using density-based clustering. In contrast, *CSSub* enables a density score to assess subspaces because the threshold

(i.e., mean of each subspace) employed is equivalent to a relative score which determines core points based on the mean value in each subspace, not an absolute density threshold across all subspaces. Therefore, *CSSub* has the ability to detect clusters with varied densities in different subspaces.

Second, the use of k-medoids or k-means in existing subspace clustering methods restricts the output to globular-shaped clusters only (e.g., PROCLUS and soft subspace clustering). Yet, *CSSub* has enabled k-medoids to be used to produce arbitrarily shaped clusters. This is possible because the clustering is performed on points defined by all subspaces under investigation, rather than the given feature space. In addition, k-medoids and k-means algorithms are unable to identify noise points. Yet, we show in Section 6.5 that *CSSub* using k-medoids is able to identify noise points.

7.3. Other Possible Refinements

We assume that only one cluster exists in each of the k subspaces detected. To examine whether the core points detected by k-medoids in a subspace have multiple clusters, density-based methods such as DBSCAN [19] can be used to separate clusters in the same subspace in the refinement stage.

It is worth mentioning that, we set τ_s in *CSSub* as the default value for every dataset, i.e., the average score value in each subspace. We found that the clustering performance on some datasets can be significantly improved when a different τ_s value is used.

The *CSSub* framework can be used to produce either non-overlapping clusters or overlapping clusters. One possible way to produce overlapping clusters is to extract multiple subspaces from the output of the k-medoids algorithm which covers most points of each cluster, rather than just the one which covers the maximum number of points in each cluster.

8. Conclusions

Grouping points by shared subspaces is a unique clustering approach which has never been attempted before, as far as we know. The approach exhibits three unique features:

1. The clustering is performed by measuring the similarity between points based on the number of subspaces they share, without examining the attribute values in the given dataset. This enables the subspace clustering to be conducted only once to produce non-redundant/non-overlapping subspace clusters. In contrast, existing algorithms produce many redundant subspace clusters from the repeated execution of clustering in a large number of subspaces because the candidate subspace selection and the clustering are tightly-coupled.

2. The approach decouples candidate subspace selection and clustering into two independent processes. As a result, it eliminates the need to have repeated runs of clustering in each of the many subspaces. The subspace selection process examines all subspaces within the limit specified for

the search, without need for a systematic search such as *Apriori*. In contrast, existing subspace clustering algorithms which have tightly-coupled processes must rely on an anti-monotonicity property to prune the search space.

3. The decoupling approach has an added advantage that it allows different types of cluster definitions to be employed easily. The time cost of the entire process is dominated by the subspace scoring function. We show that by changing the cluster definition, which has a linear time scoring function, enables the runtime of $CSSub$ to be reduced from quadratic time to linear time. A similar change is difficult, if not impossible, for existing algorithms because of the tightly coupled processes. No existing subspace clustering algorithms have the ability to use any scoring function easily, without some procedural modifications.

The use of k-medoids or k-means in existing subspace clustering methods restricts the output to globular-shaped clusters only (e.g., PROCLUS and FG-k-means). Yet, $CSSub$ has enabled k-medoids to be used to produce arbitrarily shaped clusters in noisy data. This is possible because the clustering is performed on points defined by shared subspaces, rather than the given feature space.

We show that $CSSub$ discovers subspace clusters with arbitrary shapes and that it also detects noise. $CSSub$ produces the best subspace clustering results on synthetic and real-world datasets in comparison with three state-of-the-art non-redundant subspace clustering algorithms which employ systematic search and optimisation for subspace search.

Like existing hard subspace clustering algorithms, $CSSub$ is meant for low-to-medium dimensionality only. How to extend $CSSub$ to deal with high-dimensional datasets is an interesting future work.

Acknowledgments

Most of this work was done when Ye Zhu was a Ph.D. student at Monash University, Australia. The anonymous reviewers have provided many helpful suggestions to improve this paper.

Appendix A. Algorithm to produce isolation path length

An $iForest$ consists of t $iTrees$, and each $iTree$ is built independently using a subset $\mathcal{D}$, sampled without replacement from $\pi_s(\mathbb{D})$, where $|\mathcal{D}| = \psi$. The maximum tree height is set to $h = \lceil log_2 \psi \rceil$. Note that the parameter e in $iTree$ is initialised to 0 at the beginning of the tree building process. The procedure to grow an $iTree$ is shown in Algorithm 3. We set ψ=256 and t=100 as the default parameter for building $iForest$, as suggested by Liu et al [40].

Algorithm 3 $iTree(X, e, h)$

Input: X - input data; e - current height; h - height limit.

Output: an $iTree$.

1: **if** $e \geqslant h$ OR $|X| \leqslant 1$ **then**
2: **return** $exNode\{Size \leftarrow |X|\}$;
3: **else**
4: Randomly select an attribute a from X;
5: Randomly select a split point b between min and max values of attribute a in X;
6: $X_l \leftarrow filter(X, a < b)$;
7: $X_r \leftarrow filter(X, a \geqslant b)$;
8: **return** $inNode\{\ Left\ \leftarrow iTree(X_l, e+1, h)$,
$Right \leftarrow iTree(X_r, e+1, h)$,
$SplitAttr \leftarrow a, SplitValue \leftarrow b\}$;
9: **end if**

The isolation path length score $L_s(x)$ of a point x in a subspace $s \in S$ is measured as the average path length over each $iTree$ as follows:

$$L_s(x) = \frac{1}{t}\sum_{i=1}^{t} \ell_s^i(x) \tag{A.1}$$

where $\ell_s^i(x)$ is the path length of x in tree i built in s.

The path length is the number of nodes x traverses an iTree from the root node until the traversal is terminated at a leaf node. Note that if a leaf node contains more than one points $F \subset \mathcal{D}$, the path length of that leaf node is adjusted by adding $2(\ln|F| + \gamma) - 2$, where is $\gamma \approx 0.58$ is the Euler constant [40].

Appendix B. Clustering on high-dimensional datasets

We selected 7 real-world high-dimensional datasets for clustering evaluation [9]. The properties of these datasets are presented in Table B.5.

Table B.6 shows the best F-measures of different clustering algorithms on 7 datasets. Generally, the soft subspace clustering algorithms perform the best on these high-dimensional datasets. LAC has the highest average F-measure of 0.60. Both ESSC and k-medoids have the average F-measure of 0.58,

[9] HumanActivity and Isolet datasets are from UCI Machine Learning Repository [35]. Other datasets are from scikit-feature repository [42].

Table B.5: The properties of 7 high-dimensional datasets.

Dataset	#Points	#Dimensions	#Clusters
HumanActivity	1492	561	6
Isolet	1560	617	26
Lung	203	3312	5
GLIOMA	50	4434	4
TOX_171	171	5748	4
Prostate_GE	102	5966	2
ALLAML	72	7129	2

which is higher than other subspace clustering algorithms. The hard subspace clustering algorithms $CSSub$, $CSSub_I$ and PROCLUS, which are meant for low-to-medium dimensionality only, are the worst performers on these high-dimensional datasets. This is because they only explore a tiny fraction of the total number of subspaces for a high-dimensional dataset. Note that P3C cannot get the results in reasonable time due to its exponential time complexity in dimensionality.

Table B.6: Best F-measures of different clustering algorithms on 7 datasets. The best two performers on each dataset are boldface.

	Hard subspace clustering				**Soft subspace clustering**					**Full**
Dataset	$CSSub$	$CSSub_I$	PROCLUS	P3C	EWKM	LAC	ESSC	FSC	FG-k-means	k-medoids
HumanActivity	0.36	0.43	0.31	—	0.32	0.47	0.33	0.37	**0.50**	**0.55**
Isolet	0.27	0.37	0.42	—	0.45	**0.55**	0.47	0.14	0.45	**0.64**
Lung	0.51	0.55	0.53	—	**0.78**	0.67	**0.81**	0.57	0.67	0.63
GLIOMA	0.41	0.39	0.47	—	**0.67**	0.57	**0.67**	0.50	0.51	0.54
TOX_171	0.18	0.21	0.44	—	0.47	0.49	0.45	**0.49**	**0.49**	0.42
Prostate_GE	0.57	**0.59**	0.51	—	0.58	0.57	**0.58**	0.56	0.57	0.56
ALLAML	0.55	0.44	0.69	—	0.73	**0.75**	0.72	0.71	**0.75**	0.71
Average	0.41	0.42	0.48	—	0.57	0.58	0.58	0.48	0.56	0.58

References

[1] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3rd Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[2] C. C. Aggarwal, C. K. Reddy, Data Clustering: Algorithms and Applications, Chapman and Hall/CRC Press, 2013.

[3] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, J. S. Park, Fast algorithms for projected clustering, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, SIGMOD '99, ACM, New York, NY, USA, 1999, pp. 61–72.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic subspace clustering of high dimen-

sional data for data mining applications, in: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, SIGMOD '98, ACM, New York, NY, USA, 1998, pp. 94–105.

[5] K. Sim, V. Gopalkrishnan, A. Zimek, G. Cong, A survey on enhanced subspace clustering, Data Mining and Knowledge Discovery 26 (2) (2013) 332–397.

[6] H.-P. Kriegel, P. Kröger, A. Zimek, Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering, ACM Transactions on Knowledge Discovery from Data (TKDD) 3 (1) (2009) 1.

[7] L. Parsons, E. Haque, H. Liu, Subspace clustering for high dimensional data: a review, ACM SIGKDD Explorations Newsletter 6 (1) (2004) 90–105.

[8] E. Müller, S. Günnemann, I. Assent, T. Seidl, Evaluating clustering in subspace projections of high dimensional data, Proceedings of the VLDB Endowment 2 (1) (2009) 1270–1281.

[9] A. Zimek, J. Vreeken, The blind men and the elephant: on meeting the problem of multiple truths in data from clustering and pattern mining perspectives, Machine Learning 98 (1) (2015) 121–155.

[10] S. Goil, H. Nagesh, A. Choudhary, Mafia: efficient and scalable subspace clustering for very large data sets, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999, pp. 443–452.

[11] C.-H. Cheng, A. W. Fu, Y. Zhang, Entropy-based subspace clustering for mining numerical data, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 1999, pp. 84–93.

[12] E. Müller, I. Assent, S. Günnemann, T. Seidl, Scalable density-based subspace clustering, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, ACM, New York, NY, USA, 2011, pp. 1077–1086.

[13] G. Moise, J. Sander, M. Ester, Robust projected clustering, Knowledge and Information Systems 14 (3) (2008) 273–298.

[14] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society: Series B (Statistical Methodology) (1977) 1–38.

[15] K. Kailing, H.-P. Kriegel, P. Kröger, Density-connected subspace clustering for high-dimensional data, in: Proceedings of the 2004 SIAM International Conference on Data Mining, SIAM, 2004, pp. 246–256.

[16] H.-P. Kriegel, P. Kroger, M. Renz, S. Wurst, A generic framework for efficient subspace clustering of high-dimensional data, in: Proceedings of the Fifth IEEE International Conference on Data Mining, ICDM '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 250–257.

[17] L. Kaufman, P. Rousseeuw, Clustering by means of medoids, Statistical Data Analysis Based on the L1 Norm and Related Methods (1987) 405–416.

[18] C. Bohm, K. Kailing, H.-P. Kriegel, P. Kroger, Density connected clustering with local subspace preferences, in: Proceedings of the Fourth IEEE International Conference on Data Mining, ICDM '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 27–34.

[19] M. Ester, H.-P. Kriegel, J. S, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 226–231.

[20] C. M. Procopiuc, M. Jones, P. K. Agarwal, T. Murali, A monte carlo algorithm for fast projective clustering, in: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, ACM, 2002, pp. 418–427.

[21] G. Moise, J. Sander, Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 533–541.

[22] J. A. Hartigan, M. A. Wong, Algorithm AS 136: A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28 (1) (1979) 100–108.

[23] Z. Deng, K.-S. Choi, Y. Jiang, J. Wang, S. Wang, A survey on soft subspace clustering, Information Sciences 348 (Supplement C) (2016) 84 – 106. doi:https://doi.org/10.1016/j.ins.2016.01.101.
URL http://www.sciencedirect.com/science/article/pii/S0020025516300640

[24] D. S. Modha, W. S. Spangler, Feature weighting in k-means clustering, Machine Learning 52 (3) (2003) 217–237. doi:10.1023/A:1024016609528.
URL https://doi.org/10.1023/A:1024016609528

[25] C. Domeniconi, D. Gunopulos, S. Ma, B. Yan, M. Al-Razgan, D. Papadopoulos, Locally adaptive metrics for clustering high dimensional data, Data Mining and Knowledge Discovery 14 (1) (2007) 63–97.

[26] L. Jing, M. K. Ng, J. Z. Huang, An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data, IEEE Transactions on Knowledge and Data Engineering 19 (8) (2007) 1026–1041.

[27] G. Gan, J. Wu, A convergence theorem for the fuzzy subspace clustering (fsc) algorithm, Pattern Recognition 41 (6) (2008) 1939 – 1947. doi:https://doi.org/10.1016/j.patcog.2007.11.011.
URL http://www.sciencedirect.com/science/article/pii/S0031320307005018

[28] Z. Deng, K.-S. Choi, F.-L. Chung, S. Wang, Enhanced soft subspace clustering integrating within-cluster and between-cluster information, Pattern Recognition 43 (3) (2010) 767 – 781. doi:https://doi.org/10.1016/j.patcog.2009.09.010.
URL http://www.sciencedirect.com/science/article/pii/S0031320309003483

[29] X. Chen, Y. Ye, X. Xu, J. Z. Huang, A feature group weighting method for subspace clustering of high-dimensional data, Pattern Recognition 45 (1) (2012) 434–446.

[30] J. Wang, Z. Deng, K.-S. Choi, Y. Jiang, X. Luo, F.-L. Chung, S. Wang, Distance metric learning for soft subspace clustering in composite kernel space, Pattern Recognition 52 (Supplement C) (2016) 113 – 134. doi:https://doi.org/10.1016/j.patcog.2015.10.018.
URL http://www.sciencedirect.com/science/article/pii/S0031320315003970

[31] Q. Wang, G. Chen, Fuzzy soft subspace clustering method for gene co-expression network analysis, International Journal of Machine Learning and Cybernetics 8 (4) (2017) 1157–1165. doi:10.1007/s13042-015-0486-7.
URL https://doi.org/10.1007/s13042-015-0486-7

[32] N. X. Vinh, J. Chan, S. Romano, J. Bailey, C. Leckie, K. Ramamohanarao, J. Pei, Discovering outlying aspects in large datasets, Data Mining and Knowledge Discovery 30 (6) (2016) 1520–1555.

[33] I. Assent, R. Krieger, E. Muller, T. Seidl, Dusc: dimensionality unbiased subspace clustering, in: Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, IEEE, 2007, pp. 409–414.

[34] P. Jaccard, The distribution of the flora in the alpine zone., New phytologist 11 (2) (1912) 37–50.

[35] M. Lichman, UCI Machine Learning Repository (2013).
URL http://archive.ics.uci.edu/ml

[36] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics Quarterly 2 (1-2) (1955) 83–97.

[37] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[38] L. Parsons, E. Haque, H. Liu, Subspace clustering for high dimensional data: A review, SIGKDD Explor. Newsl. 6 (1) (2004) 90–105. doi:10.1145/1007730.1007731.
URL http://doi.acm.org/10.1145/1007730.1007731

[39] L. O. Jimenez, D. A. Landgrebe, Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 28 (1) (1998) 39–54. doi:10.1109/5326.661089.

[40] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, ACM Transactions on Knowledge Discovery from Data (TKDD) 6 (1) (2012) 3:1–3:39.

[41] S. J. Russell, J. Stuart, Artificial Intelligence: A Modern Approach (2nd Edition), Pearson Education, 2003.

[42] J. Li, K. Cheng, S. Wang, F. Morstatter, T. Robert, J. Tang, H. Liu, Feature selection: A data perspective, arXiv:1601.07996.