Subspace Clustering of High Dimensional and Streaming Data
Seminar Multimedia Retrieval and Data Mining

István Sárándi

Data Management and Data Exploration Group
RWTH Aachen University
Germany
istvan.sarandi@rwth-aachen.de

Abstract. High-dimensional and streaming data have become widely available in recent years. In the meantime, new data mining algorithms have also been proposed that can efficiently and effectively process these types of data. In this seminar paper we will discuss the requirements for these scenarios and look at various algorthmic solutions to the clustering problem. A particularly active reasearch topic in high-dimensional data mining is subspace clustering. CLIQUE is a prominent example of subspace clustering algorithms and the main part of the paper gives a detailed explanation of its steps. Streaming data can be efficiently processed with the microcluster approach. CluStream, DenStream and their alternatives are also discussed, as well as their applicability together with CLIQUE. Finally, these combinations of CluStream with CLIQUE and DenStream with CLIQUE are evaluated on a synthetic and a real dataset using the SubspaceMOA framework.

1 Introduction

Due to the rapidly increasing amount of high-dimensional and streaming data, it is becoming more and more important to be able to analyze and make sense of this kind of data efficiently. Clustering is one of the basic data mining problems with far-reaching applications and it has been studied extensively in the past (e.g. see [17] for an introduction). In this paper, we will look at how the clustering problem can be approached in the context of high-dimensional and streaming data, what type of new challenges arise and how they can be dealt with.

A common approach in the high-dimensional scenario is subspace clustering, while streaming data has been successfully analyzed using microcluster-based algorithms. In a large portion of the paper we will look at CLIQUE, a subspace clustering algorithm; followed by two microcluster approaches, CluStream and DenStream. Then we will discuss how these algorithms can be combined with each other to yield a solution that can operate is high-dimensional and streaming contexts at the same time. In the rest of the introduction section, we briefly introduce the clustering problem and the characteristics of high-dimensionality and streaming.

1.1 The clustering problem and its applications

According to the definition by Jain et al.[17], clustering is the organization of a collection of patterns into clusters based on similarity. The motivation for grouping data into clusters may be to gain new insights about the data (e.g. split the customer database to previously unknown groups for targeted marketing), to compress or summarize the data for efficiency (e.g. microcluster approaches for streaming data) or as a substitute for classification when no ground-truth labels are available (e.g. some applications in image segmentation).

There is no single formal definition of clustering. One of the challenges in data clustering is to find a reasonable formalization of the problem itself. A usual formulation is based on a pairwise similarity or distance function. The goal is then to find clusters such that objects in the same cluster are similar to each other, while objects in different clusters are dissimilar. Simple and popular clustering algorithms based on this idea are k-means[20] and related methods (k-median, k-medoid etc.).

A different formulation is based on a transitivity principle. Objects that are dissimilar regarding a pairwise comparison may still be included in the same cluster, provided that there is a sequence of intermediate objects, each similar enough to the previous one (pairwise). This enables the detection of clusters of arbitrary shape. A prominent example for such an algorithm is DBSCAN[10].

Comparison with classification Clustering is sometimes called unsuperwised classification. Unsuperwised means that the input of the clustering algorithm are only the data objects themselves. By contrast, supervised classification has two phases. In the first, so called training phase, the input of the algorithm also contains the true class to which each data point belongs (labeled data). The algorithm must learn how the class can be predicted from the data. In the second, so called test phase, some new, unlabeled data is fed into the algorithm and it tries to predict the class to which the objects belong. In this paper we will only discuss unsuperwised clustering.

1.2 High-dimensional data

Computational processing power and storage has exponentially grown in the last decades and this makes it possible to create huge collections of high-dimensional data[19]. A notoriously data-intensive field is computational genomics. Other applications include text mining and analysis of user behavior[19]. To analyze such data, new data mining methods must be developed that scale well with the increased data dimensionality and quantity. However, the problem is not only about computational efficiency. The ideas behind traditional data mining techniques need to be re-evaluated as well.

In particular, traditional distance measures become meaningless in the presence of noise in high-dimensional spaces. If we assume independent and identically distributed (i.i.d.) data points and attribute values, the distances from

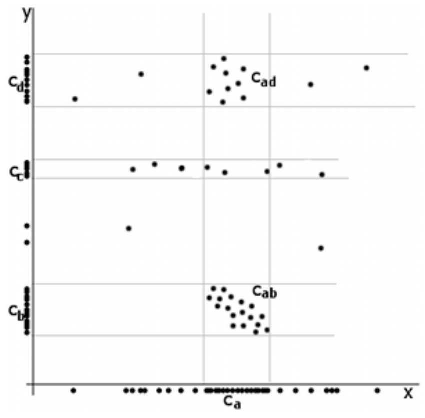

Fig. 1. Illustration of subspace clusters in 2 dimensions[26]

a query point to other points will become indistinguishable with growing dimensionality[5]. The i.i.d. assumptions do not hold for clustered data, but since clusters are usually lower-dimensional, the remaining space can be assumed to be noisy and i.i.d., resulting in a similar effect: the low-dimensional cluster is hidden in the vast high-dimensional space. Global dimensionality reduction (such as principal component analysis) is not helpful here, since each cluster has its own relevant subspace. Restricting subspaces to axis-parallel projections is a better alternative to reduce the dimensionality. Figure 1 shows the principle of axis-parallel subspace clusters in two dimensions for easy visualization.

This approach works because we know that the attributes in the dataset are not arbitrary directions in the high-dimensional space. We can assume that, in reality, many attribute groups are highly independent from other attributes and thus the true clusters will tend to use only some of the attributes. Thus we can avoid the irrelevant dimensions' contribution to noise[23].

Looking for clusters in oblique projections (e.g. arbitrary linear combinations of the dimensions) would be problematic in many ways. It needs more computational power, it can result in more false positives (out of the infinite possibilities to construct oblique projections in high dimensional data, many spurious clusters would emerge by chance), and such projections are hard to interpret.

Kriegel et al. survey high-dimensional clustering algorithms in a systematic way[19]. The two basic types of problem formulations and algorithms are *subspace clustering* and *projected clustering*. In the former, a data object may be contained in multiple clusters, while the latter partitions the dataset to clusters without overlap. This seminar report is centered around CLIQUE[3], a subspace clustering algorithm based on a regular grid subdivision of the data space and testing it in the context of stream clustering. A detailed description is given in Section 2.

1.3 Streaming data

Communication networks have also seen important developments in recent years. Increased data transmittion speed and connectivity allows to continuously send large volumes of data, such as measurements from sensor networks. Such continuously flowing data is referred to as streaming data. Intelligent analysis of streaming data must cope with new requirements. Gaber[11] and Gaber et al.[12] give a review of what these requirements are and what developments have taken place.

Dealing with streaming data is more than "just" efficiently processing large volumes of incoming data. The characteristics of the data stream evolve over time. The user is most interested in the current characteristics, so older parts of the data become gradually less relevant. The user may also want to explicitly analyze the temporal evolution of the data distribution (e.g. movement, growth, merger or splitting of clusters).

Restricting analysis to recent samples can be done by windowing. Such algorithms are ADWIN and ADWIN2 ('adaptive windowing') that adaptively set the window length to cover samples since the last major change in data distribution[6]. Other approaches use a tilted (weighted) time-window (e.g. HP-Stream[2]) or sample the data, storing more samples for recent data and less for older data (e.g. Clusmaster [9]). All these are types of *aging techniques*.

Ideally, we would incrementally update the results of our complex mining algorithms after the arrival of every new object (adjusted with aging), to reflect the current distribution and structure of the data stream. Unfortunately, this is computationally infeasible. Therefore a two-phase approach was introduced by Aggarwal et al.[1]. In the online phase, we incrementally maintain some reduced amount of information that is most crucial to determine the current characteristics of the data distribution. To reduce memory requirements, we need to do this at a coarser granularity than individual samples. A popular solution is to summarize data points into microclusters and to store only incrementally updatable statistics about them. Then, in the offline phase, the computationally more intensive clustering algorithm (e.g. CLIQUE) is used on these microclusters.

Such microclustering approaches are CluStream[1] and DenStream[7]. They can be used together with any offline clustering algorithm, although the authors suggest using k-means and DBSCAN in their respective papers. Section 3 gives more details about CluStream and DenStream.

D-Stream[8], introduced by Chen et al., is also a two-phase stream mining algorithm. It is based on a grid subdivision of the data space and stores information about how the number of objects in each cell evolves over time. In contrast to CluStream and DenStream, D-Stream has its own offline algorithm that relies on internal state of grid cells. Hence, without major modifications to its framework, it is not usable with custom offline algorithms, and therefore we will not discuss it further in this paper.

1.4 Structure of the paper

Section 2 will introduce CLIQUE and explain its steps in details with clear pseudocodes. At the end of the section, CLIQUE is compared with newer subspace algorithms. Section 3 discusses the microcluster algorithms CluStream and DenStream. In Section 4, we will evaluate CLIQUE in combination with CluStream and DenStream on a real and a synthetic dataset.

2 The CLIQUE algorithm

CLIQUE[3] is a popular subspace clustering algorithm proposed by Agrawal et al. It was developed to fulfil the following requirements:

i) **Effective treatment of high-dimensionality:** As discussed in the introduction, high-dimensional datasets do not have meaningful clusters in the whole data space. For this reason, CLIQUE looks for clusters in subspaces.
ii) **Interpretability of results:** Clustering is used in a variety of scenarios. CLIQUE's scenario is exploratory analysis, i.e. a human user wants to gain insight about the structure of the data. One way of making data easy to interpret is visualization, but it is only effective up to 3 or 4 dimensions. CLIQUE solves interpretability by restricting subspaces to axis-parallel projections, excluding oblique projections. The authors note that even a linear combination of two attributes would create a concept that users can not interpret.
iii) **Scalability:** The algorithm should be fast and scale to large databases and high-dimensionality. This is achieved by a clever way of searching through subspaces.
iv) **Usability:** The result should be independent of the order of the input data objects and it should not be assumed that they were generated according to some special distribution (e.g. Gaussian). In particular, CLIQUE can return arbitrary shaped clusters.

An informal overview of CLIQUE is the following. We subdivide the input space with a regular grid by partitioning each dimension to ξ equal-length intervals. Then, in each subspace, we look for grid units (cells) that contain at least τ data objects (these are called dense units). A cluster is a collection of connected dense units that lie in the same subspace. After clusters are found, we generate short DNF descriptions for them. Thus CLIQUE has three main steps:

1) **Finding dense units in subspaces:** efficient Apriori-like bottom-up search on the subspace lattice with heuristic pruning
2) **Forming clusters:** connected component labeling by depth-first search
3) **Generating short DNF descriptions for clusters:** greedy growth of hyperrectangles followed by greedy redundancy elimination

As we can see, CLIQUE uses several, largely unrelated heuristic components to do clustering. After this intuitive description, let us consider the task and terminology more formally. We will build upon the notions of Agrawal et al. but also extend it to more detailed notation.

2.1 Formal definitions

A **dimension** (or attribute) A is a named range of real numbers

$$A = (A.name, [A.min, A.max]).$$

Let $\mathcal{A} = \{A_1, ..., A_D\}$ be the dimensions in our D-dimensional input dataset. The input space is

$$\mathcal{V} = [A_1.min, A_1.max] \times ... \times [A_D.min, A_D.max].$$

A **subspace** is defined as a set of dimension indices, so the set of subspaces is $\mathcal{S} = 2^{\{1,...,D\}}$. The set of k-dimensional subspaces is denoted by

$$\mathcal{S}_k = \big\{ S \in \mathcal{S} \bigm| |S| = k \big\}$$

The set of N input data objects is

$$V = \{\mathbf{v}_1, ..., \mathbf{v}_N\} \subset \mathcal{V}.$$

Each dimension $A \in \mathcal{A}$ is discretized to ξ equal, non-overlapping intervals. The size of each interval (bin) is $A.step = (A.max - A.min)/\xi$. The resulting discretized **grid** for a k-dimensional subspace is

$$\mathcal{G}_k = \{0, ..., \xi - 1\}^k .$$

The full grid is $\mathcal{G}_D$. A **unit** (or subspace cell) is defined by its subspace and its coordinates in the subspace grid, so the set of k-dimensional units is:

$$\mathcal{U}_k = \mathcal{S}_k \times \mathcal{G}_k.$$

The coordinates are always specified in the order of ascending dimension indices. The **selectivity** of a unit is defined as the number of data objects contained in it:

$$sel(u) = \big|\big\{ \mathbf{v} \in V \bigm| contains(u, \mathbf{v}) \big\}\big| ,$$

where *contains* means for a unit $u = (\{d_1, ..., d_k\}, (g_{d_1}, ..., g_{d_k}))$

$$contains(u, \mathbf{v}) \iff \forall i \in \{1, ..., k\} : \left\lfloor \frac{v_{d_i} - A_{d_i}.min}{A_{d_i}.step} \right\rfloor = g_{d_i} \tag{1}$$

A unit is **dense** iff $sel(u) \geq \tau$. A **cluster** is a pair of a subspace and a set of grid coordinates, representing a maximal set of connected dense units in that subspace. In a k-dimensional subspace $S = \{d_1, ..., d_k\}$, a cluster has the form $C = (S, G), G \subseteq \mathcal{G}_k$. Such a pair C is a cluster iff all of the following hold:

$$\forall \mathbf{g} \in G : sel((S, \mathbf{g})) \geq \tau$$

$$\forall \mathbf{g}, \mathbf{g}' \in G : connected((S, \mathbf{g}), (S, \mathbf{g}'))$$

$$\nexists \mathbf{g} \in G, \mathbf{g}' \in \mathcal{G}_k \setminus G : connected((S, \mathbf{g}), (S, \mathbf{g}')),$$

where *connected* is the transitive closure of the has-common-face relation among dense units. For two units $u = (S, (g_{d_1}, ..., g_{d_k}))$ and $u' = (S, (g'_{d_1}, ..., g'_{d_k}))$

$$hasCommonFace(u, u') \iff \exists i \in S : \big(|g_i - g'_i| = 1 \land \forall j \in S \setminus \{i\} : g_j = g'_j\big) \tag{2}$$

A k-dimensional hyperrectangular **region** with upper and lower bound coordinates $\mathbf{min}, \mathbf{max} \in \mathcal{G}_k$ is denoted as

$$Rect(\mathbf{min}, \mathbf{max}) = \big\{\mathbf{g} \in \mathcal{G}_k \bigm| \forall i \in \{1, ..., k\} : min_i \leq g_i \leq max_i\big\}$$

A **covering** for a cluster $C = (S, G)$ is a set of hyperrectangular regions $\mathcal{R} = \{R_1, ..., R_m\}$ such that

$$\bigcup_{i=1}^{m} R_i = G$$

Finally the goal of CLIQUE can be stated precisely. The algorithm should find all clusters $C = (S, G)$ in the input data, and provide a (small) covering for each.

Let us now examine each of the three main steps of CLIQUE in more detail. We will also see a number of pseudocode fragments for the parts of the algorithm that are not clarified exactly in Agrawal et al.'s paper.

2.2 Finding dense units in subspaces

The efficient bottom-up search is based on the monotonicity lemma: If a k-dimensional unit is dense, then all its $(k-1)$-dimensional projection units are also dense. It is obviously true, since a projection contains all objects of the original unit and possibly more, so it must also surpass the τ threshold. Thus, if we already know the dense units in the $(k-1)$-dimensional subspaces, we only need to examine a reduced number of k-dimensional units. This principle results in an iterative algorithm beginning with finding 1-dimensional subspaces and increasing the dimensionality stepwise (Alg. 1).

We find 1-dimensional dense units by building a histogram for each dimension during a single pass over the whole database.

Now suppose that all $(k-1)$-dimensional dense units have been determined and we want to find dense units in k-dimensional subspaces. Any dense units in such a subspace $S = \{d_1, ..., d_k\}$, $(d_1 < ... < d_k)$ must have dense projections in all $(k-1)$-dimensional subspaces $S' \subset S$. There are k such subspaces, so the monotonicity lemma yields k constraints. A naive solution would enumerate all units in S and check if all k constraints are satisfied for them. CLIQUE uses a much faster way: we first explicitly generate all k-dimensional units that satisfy two of their k constraints and then check each generated unit for the remaining $k-2$ constraints. The two constraints correspond to the subspaces $S'_1 = \{d_1, ..., d_{k-2}, d_{k-1}\}$ and $S'_2 = \{d_1, ..., d_{k-2}, d_k\}$.

In the generation procedure (Alg. 2), we look for pairs of $(k-1)$-dimensional dense units (Alg. 2, Line 3) that share their first $k-2$ dimensions but not the

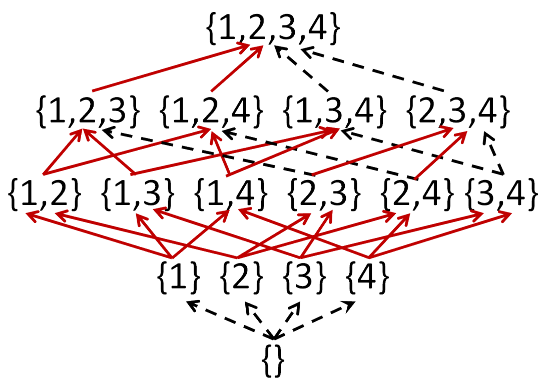

Fig. 2. Subspace lattice. The solid red arrows show which subspaces are used in the JOIN procedure to create the candidate dense units.

$(k-1)^{\text{th}}$ and furthermore, their values along the first $k-2$ dimensions are identical as well (Alg. 2, Line 6). Each such pair of dense units is then combined to yield a k-dimensional candidate that certainly satisfies two of its constraints, since two of its projections are the units from which it was combined. Then the remaining constraints are also checked and if any of them is violated, the candidate is discarded (Alg. 1, Line 9). The surviving candidate units are then checked for their selectivity in a single pass through the data (Alg. 1, Line 25), where for each candidate unit we count how many times data objects fall into them. Units whose counters reach the threshold τ are exactly the dense units in k-dimensional subspaces. Figure 2 illustrates which subspaces are joined to create higher-dimensional candidates in a toy example of 4 dimensions.

This algorithm would still have exponential runtime in the highest dimensionality of any dense unit, rendering it infeasible for high-dimensional data. Therefore, the authors of CLIQUE came up with a heuristic speedup, which at the same time makes the algorithm no longer exact (i.e. some dense units may be missed). Before generating k-dimensional dense unit candidates from the $(k-1)$-dimensional dense units, the $(k-1)$-dimensional dense units that lie in 'uninteresting' subspaces are pruned (Alg. 3, called in Alg. 1 at Line 34). To decide which subspaces are uninteresting, the coverage is calculated for each subspace, defined as the number of data objects that lie in some dense unit of the given subspace.

$$cov(S) = \left\{ \mathbf{v} \in V \;\middle|\; \exists \mathbf{g} : sel((S,\mathbf{g})) \geq \tau \wedge contains((S,\mathbf{g}),\mathbf{v}) \right\}$$

The larger the coverage, the more likely it is that there are dense units in its superspaces. This is only a heuristic measure, other measures could also be invented.

Although not stated in the paper, the coverage of a subspace S has the useful property that it is equal to the sum of the selectivities of the dense units in subspace S.

Algorithm 1 Apriori-like bottom-up algorithm for finding dense units

```
1: function FindDenseUnits
2:     make one pass over $V$ and build a histogram $hist_i$ for each dimension $A_i$
3:     $Den_1 \leftarrow \{(\{i\},(g)) \mid hist_i[g] \geq \tau\}$
4:     // dense units in 1D

5:     for $k \leftarrow 2; k \leq D; k \leftarrow k+1$ do
6:         $Cand \leftarrow$ Join$(Den_{k-1})$
7:         // See Alg. 2

8:         // check all projections to $(k-1)$ dimensions
9:         for all $(S, \mathbf{g}) \in Cand$ do
10:            $(d_1, ..., d_k) \leftarrow sorted(S)$
11:            for all $d \in \{d_1, ..., d_{k-2}\}$ do
12:                $u_{proj} \leftarrow (S \setminus d, (g_1, ..., g_{d-1}, g_{d+1}, ..., g_k))$
13:                if $u_{proj} \notin Den_{k-1}$ then
14:                    // $u$ has a non-dense projection
15:                    $Cand \leftarrow Cand \setminus u$
16:                    continue loop of line 9
17:                end if
18:            end for
19:        end for

20:        for all $u \in Cand$ do
21:            $selectivity[u] \leftarrow 0$
22:            // initialize frequency counters for candidate cells
23:        end for

24:        for all $\mathbf{v} \in V$ do
25:            // pass over the data
26:            for all $u \in Cand$ do
27:                if $contains(u, \mathbf{v})$ then
28:                    // See Eq. 1 for the definition of contains
29:                    $selectivity[u] \leftarrow selectivity[u] + 1$
30:                end if
31:            end for
32:        end for

33:        $Den_k \leftarrow \{u \mid selectivity[u] \geq \tau\}$
34:        $Den_k \leftarrow$ PruneMDL$(Den_k, selectivity)$
35:        // See Alg. 3
36:        if $|Den_k| < 2$ then
37:            break
38:        end if
39:    end for

40:    return $\bigcup_{k=1}^{D} Den_k$
41: end function
```

Algorithm 2 Algorithm for creating dense unit candidates

```
1: function Join($Den_{k-1}$)
2:     $Cand \leftarrow \emptyset$
3:     for all $((S, \mathbf{g}), (S', \mathbf{g}')) \in Den_{k-1} \times Den_{k-1}$ do
4:         $(d_1, ..., d_{k-1}) \leftarrow sorted(S)$
5:         $(d'_1, ..., d'_{k-1}) \leftarrow sorted(S')$
6:         if $(\forall i \in \{1, ..., k-2\} : d_i = d'_i \wedge g_i = g'_i) \wedge d_{k-1} < d'_{k-1}$ then

7:             // if same projection to the subspace of their first $(k-2)$ dimensions
8:             $c \leftarrow (S \cup S', (g_1, ..., g_{k-2}, g_{k-1}, g'_{k-1}))$
9:             $Cand \leftarrow Cand \cup \{c\}$
10:        end if
11:    end for

12:    return $Cand$
13: end function
```

$$cov(S) = \sum_{u \in \left\{ (S,\mathbf{g}) \middle| sel((S,\mathbf{g})) \geq \tau \right\}} sel(u)$$

Hence, coverages can be computed without an additional database scan, since the selectivities have been computed prior to pruning.

Subspaces are sorted according to decreasing coverage, yielding $S_1, ..., S_n$. We choose a cutting point i and keep the subspaces $\{S_1, ..., S_i\} = \sigma_{keep}$ while $\{S_i + 1, ..., S_n\} = \sigma_{prune}$ are pruned (discarded). The cutting point is determined by a minimum description length (MDL) scheme. We aim to minimize the number of bits that would be required to store the following:

i) Mean of the coverage values of kept subspaces (rounded up to the next integer):

$$\mu_{keep}(i) = \left\lceil \frac{\sum_{j=1}^{i} cov(S_j)}{i} \right\rceil$$

ii) Same for pruned subspaces:

$$\mu_{prune}(i) = \left\lceil \frac{\sum_{j=i+1}^{i} cov(S_j)}{n-i} \right\rceil$$

iii) In-group absolute deviations

$$|cov(S_j) - \mu_{keep}(i)| \quad \forall j \in \{1, ..., i\}$$
$$|cov(S_j) - \mu_{prune}(i)| \quad \forall j \in \{i+1, ..., n\}$$

The number of bits required is the sum of the base-two logarithms, denoted by $CL(i)$ (for 'code length'). Each possible i is evaluated and the one with minimum $CL(i)$ is chosen.

$$
\begin{aligned}
CL(i) = &\log_2 \mu_{keep}(i) + \sum_{j=1}^{i} \log_2 \left| cov(S_j) - \mu_{keep}(i) \right| + \\
&\log_2 \mu_{prune}(i) + \sum_{j=i+1}^{n} \log_2 \left| cov(S_j) - \mu_{prune}(i) \right| ,
\end{aligned}
$$

Algorithm 3 Pruning of dense units for efficiency

```
1: function PRUNEMDL($Den, selectivity$)
2:     $\sigma \leftarrow \{S \mid \exists \mathbf{g} : (S, \mathbf{g}) \in Den\}$

3:     for all $S \in \sigma$ do
4:         $cov[S] \leftarrow \sum_{(S,\mathbf{g}) \in Den} selectivity[(S, \mathbf{g})]$
5:         // Sum of selectivities of dense units
6:     end for

7:     $n \leftarrow |\sigma|$
8:     $(S_1, ..., S_n) \leftarrow$ sort $\sigma$ by $cov[.]$ to descending order
9:     $i^* \leftarrow \arg\min_{1<i<n} CL(i)$
10:    $Den' \leftarrow \{(S, \mathbf{g}) \in Den \mid S \in \{S_1, ..., S_{i^*}\}\}$
11:    return $Den'$
12: end function
```

By this modification of the bottom-up algorithm, we get a feasible method to determine the dense units (or at least most of them, if the heuristic works well). The next step is to find connected dense units in each subspace to form clusters.

2.3 Forming clusters

Having determined the dense units, now we need to unite them to form clusters. This is done separately in each subspace S, in which we have found some dense units. We perform connected component labeling by depth first search (DFS) in the following graph: nodes are the dense units found in S and two nodes are connected by an edge if their corresponding units have a common face (see Eq. 2 about the "has common face" relation).

2.4 Generating short DNF descriptions for clusters

Now we have determined which sets of dense units form clusters. In the last step, we take each cluster and generate a short definition for it to make the results easier to interpret. CLIQUE uses disjunctive normal form (DNF) expressions to generate a description for each cluster. A disjunctive normal form is the

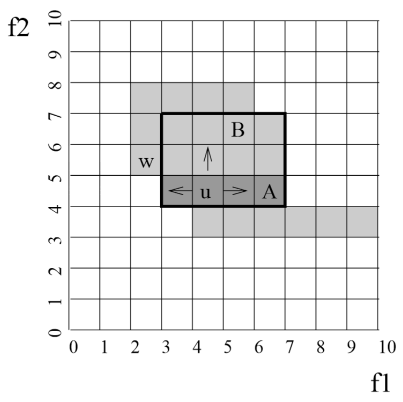

Fig. 3. Illustration of the greedy covering procedure (from [3])

OR-connection of AND-clauses, e.g. $(A \wedge B \wedge C) \vee (D \wedge E) \vee (F \wedge G \wedge H)$ is a DNF expression. Boolean variables denoted by letters in this example correspond to attribute comparisions (e.g. $shoeSize < 44$) in the CLIQUE ouput. Each clause represents one hyperrectangular region and the union of the regions covers exactly the units of the cluster. The goal is to generate a short description, that is, use as few hyperrectangles (clauses) as possible but still cover all units of the cluster and no others.

This is a variant of the set cover problem and it is known to be NP-hard. Therefore the authors of CLIQUE propose a greedy heuristic algorithm to create a practical solution. An initial unit is picked and a region is grown greedily around it to maximal size along each dimension. This is repeated until all units of the cluster are covered (Alg. 4 and Fig. 3).

This procedure may end up with redundant regions, whose units are all covered by some other regions. To eliminate these redundancies, Agrawal et al. propose a greedy removal heuristic. We sort the regions according to the number of contained units and successively remove redundant regions (Alg. 5).

The resulting covering can be trivially converted to a DNF expression. The description of the algorithm is now complete, we have determined the subspace clusters and the corresponding DNF descriptions.

2.5 Comparison with related algorithms

There has been considerable research interest in subspace clustering during the time since 1998, when CLIQUE first appeared. Several shortcomings of CLIQUE have since been improved in newer algorithms. MAFIA[13] uses an adaptively spaced grid for subdivision, based on the distribution of the data. This reduces the required number of units, since large homogeneous regions are stored in a single block. Because of this speedup, the MDL-based pruning of subspaces is no longer required in MAFIA. Furthermore, CLIQUE may miss the edges of

Algorithm 4 The greedy region covering algorithm

```
 1: function GreedyCover($C$)
 2:     $(S, G) \leftarrow C$
 3:     $uncovered \leftarrow G$
 4:     $\mathcal{R} \leftarrow \emptyset$
 5:     while $uncovered \neq \emptyset$ do
 6:         pick $\mathbf{g} \in uncovered$
 7:         $R \leftarrow Rect(min : \mathbf{g}, max : \mathbf{g})$
 8:         for all $d \in S$ in random order do
 9:             $R.min_d \leftarrow \min \{x \mid \forall \mathbf{g}' \in Rect(R.min[d \leftarrow x], R.max) : \mathbf{g}' \in G\}$
10:             $R.max_d \leftarrow \max \{x \mid \forall \mathbf{g}' \in Rect(R.min, R.max[d \leftarrow x]) : \mathbf{g}' \in G\}$
11:         end for
12:         $uncovered \leftarrow uncovered \setminus R$
13:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$
14:     end while
15:     return $\mathcal{R}$
16: end function
```

Algorithm 5 The greedy redundancy elimination algorithm

```
1: function GreedyEliminateRedundancy($\mathcal{R}$)
2:     for all $R \in \mathcal{R}$ in ascending order by size do
3:         if $\forall \mathbf{g} \in R : \exists R' \in \mathcal{R} \setminus \{R\} : \mathbf{g} \in R'$ then
4:             $\mathcal{R} \leftarrow \mathcal{R} \setminus \{R\}$
5:         end if
6:     end for
7:     return $\mathcal{R}$
8: end function
```

clusters, since the grid might cross them in an unfortunate way. MAFIA doesn't have this issue, since the grid moves adaptively to the border of the cluster.

SCHISM[24] extends CLIQUE by varying the threshold τ (selectivity required for a dense unit) depending on subspace dimensionality. The rationale is that higher-dimensional units have a lower probability of attaining high-density, since there are exponentially more units for increasing dimensionalty. The threshold is derived using a combination of statistical hypothesis testing and the Chernoff–Hoeffding Bound theorem.

CLIQUE has an additional issue, that for each high-dimensional cluster containing data points V in subspace S, some superset $V' \supset V$ will also form a cluster in every possible projected subspace $S' \subset S$. In short, the clusters contain redundant information. The so called *non-redundant subspace clustering* algorithms have been developed to address this problem (e.g. INSCY[4], RESCU[22], OSCLU[14] and STATPC[21]).

SUBCLU[18] was proposed based on DBSCAN and using its principles of density-connectivity. Similar to CLIQUE, it also uses an Apriori-like bottom-up search on the subspace lattice, but it uses no grid subdivision. The lack of grid makes SUBCLU more accurate but much slower than CLIQUE.

3 Microcluster approaches

As explained in the introduction, streaming data is becoming important. To apply CLIQUE in such a scenario, the two-phase approach can be used with microclustering in the online phase.

3.1 CluStream

In this section we will discuss the CluStream algorithm for microcluster management.

In CluStream[1], each incoming data object $\mathbf{v} \in \mathcal{V}$ is added to a microcluster by updating the summary statistics stored in it. Note that the data object itself is discarded! A fixed number of q microclusters are used and the statistics maintained for each microcluster are the following:

i) N: number of data objects that were merged into the microcluster.
ii) $LS_d = \sum_{i=1}^{N} v_{i,d} \quad \forall d \in \{1, ..., D\}$ (linear sum)
iii) $SS_d = \sum_{i=1}^{N} v_{i,d}^2 \quad \forall d \in \{1, ..., D\}$ (squared sum)
iv) $LST = \sum_{i=1}^{N} t_i$ (linear sum of the timestamps)
v) $SST = \sum_{i=1}^{N} t_i^2$ (squared sum of the timestamps)
vi) (ids: list of identifiers of previous clusters that were unified into this one, as we will see later.)

Since each of the statistics are sums over the contained data objects, they can be incrementally updated. We can compute derived statistics from these values, such as mean (μ) and variance (σ). By storing information about the

Algorithm 6 Online part of the CluStream algorithm

```
1: procedure CluStreamOnline($InitNumber, q, \alpha, m, \delta$)
2:     // Initialize microclusters
3:     $\mathcal{M} \leftarrow$ OfflineCluster(first $InitNumber$ samples from the stream)
4:     for each new sample $\mathbf{v}$ with timestamp $t$ do
5:         $M^* \leftarrow \arg\min_{M \in \mathcal{M}} dist(\mathbf{v}, \mu(M))$
6:         if $dist(\mathbf{v}, \mu(M^*)) \leq \alpha\sigma(M^*)$ then
7:             update statistics of $M^*$ by adding $\mathbf{v}$
8:         else
9:             $M_{new} \leftarrow$ new microcluster containing $\mathbf{v}$
10:            for all $M \in \mathcal{M}$ do
11:                // relevance stamp is at the $\frac{m}{2N(M)}$th percentile of the timestamps
12:                $t_r[M] = \mu_{time}(M) + \sigma_{time}(M) \cdot \Phi^{-1}\left(\frac{m}{2N(M)}\right)$
13:            end for
14:            $M_{oldest} \leftarrow \arg\min_{M \in \mathcal{M}} t_r[M]$
15:            if $t_r[M_{oldest}] \leq \delta$ then
16:                $\mathcal{M} \leftarrow \mathcal{M} \setminus \{M_{old}\}$
17:            else
18:                merge the two $M_1, M_2 \in \mathcal{M}$ with minimal $dist(\mu(M_1), \mu(M_2))$
19:            end if
20:            $\mathcal{M} \leftarrow \mathcal{M} \cup \{M_{new}\}$
21:        end if
22:    end for
23: end procedure
```

timestamps, we can delete old microclusters that are no longer relevant. The sketch of the algorithm is given in Algorithm 6.

This online algorithm is constantly running. At regular time intervals, snapshots are saved containing all the statistics stored for the q microclusters at the given time. These snapshots are stored in a data structure called the *pyramidal time frame*. It stores less of the old snapshots and more of the recent ones. It guarantees that if we want to retrieve a snapshot of time t at current time t_c, there will be a stored snapshot in the interval $[t - \beta \cdot (t_c - t), t]$, where β depends on parameterization. The data structure requires $\mathcal{O}(\beta^{-1} \log(T))$ memory to store T snapshots. Accuracy and memory requirement can be improved on the cost of the other.

The offline algorithm is executed at user request. We only want to consider recent samples. Therefore, the contribution of old samples must be removed from the current microclusters. This is done by first retrieving a microclustering snapshot at some defined time in the past $t = t_c - t_0$. Assume that the pyramidal time frame returns a snapshot at time $t' \leq t$. Then we look for microclusters in the old snapshot that survived to the current time (possibly merged). This can be done using the *ids* list, which is unified at each merger. Since the microcluster statistics are additive, simple subtraction will give the information about samples since t'. These subtracted statistics give rise to a new set of "recent" microclusters that describe the data since t'.

Next, the recent microclusters are used as input to an offline clustering algorithm. The algorithm needs to be adapted to work with microclusters instead of data objects. For example, k-means would need adjustment at the mean computations, to take into account the different number of data objects contained in each microcluster.

3.2 DenStream

Cao et al. proposed another microclustering algorithm, called DenStream[7]. Similar to CluStream, this approach also uses summary statistics, but in DenStream they are time-weighted. Exponential weight decay is applied, due to its temporal multiplicity property. Another difference is that DenStream uses multiple kinds of microclusters: c-microclusters (core), p-microclusters (potential core) and o-microclusters (outlier). Only p- and o-microclusters are maintained in the online phase. The maintained statistics for each microcluster are:

i) $N^t = \sum_{i=1}^{N} w(t_i, t_{current})$ (sum of time-weights)
ii) $LS_d^t = \sum_{i=1}^{N} w(t_i, t_{current}) v_{i,d} \quad \forall d \in \{1, ..., D\}$ (time-weighted linear sum)
iii) $SS_d^t = \sum_{i=1}^{N} w(t_i, t_{current}) v_{i,d}^2 \quad \forall d \in \{1, ..., D\}$ (time-weighted squared sum)

For o-microclusters the time of their creation is also stored. Let us refer to microclusters whose $N^t \geq \tau$ as dense microclusters. The difference between the meaning of p and o-microclusters is that p-microclusters must be dense.

The algorithm is initialized by taking $InitNumber$ data objects from the stream and sequentially creating p-microclusters for each object $\mathbf{v}$, whose ϵ

neighborhood contains at least τ objects (these neighbors are removed before looking for the next object $\mathbf{v}$).

For each new incoming data object, we try to merge it into an existing p-microcluster. If the nearest p-microcluster would keep a variance lower than ϵ after the merger, then we merge the object into that microcluster. Otherwise the same is performed for the nearest o-microcluster. If we could merge it to an o-microcluster, then we check whether it has now become dense. If so, it is promoted to p-microcluster status. If we cannot merge to any o-microcluster, then a new outlier microcluster must be created (with the current timestamp stored as creation time). Pseudocodes can be found in [7].

In contrast to CluStream, the number of microclusters is not fixed in DenStream. In order to avoid having too many microclusters, DenStream periodically deletes p-microclusters, whose N^t falls below the τ threshold and o-microclusters whose N^t haven't reached the τ threshold in the last T timesteps.

3.3 Combining CLIQUE and microclusters

To handle stream data, CLIQUE has to be adapted to work on microclusters. There are multiple possibilities to do this. The most straightforward way is to generate several data objects from each microcluster and execute CLIQUE on the generated objects. This can be done by the following resampling approach. Assume a microcluster with size N, where size refers to the possibly time-weighted number of data object represented by the microcluster. We generate $\alpha \cdot N$ samples from the multivariate normal distribution with mean and variance parameters computed from the incremental microcluster statistics. We repeat the process for all microclusters.

We should note that this is not necessarily the best approach. CLIQUE only needs to know the selectivity of each grid unit, but does not need the actual data objects. This fact could be used to devise an alternative way to adapt CLIQUE to the microcluster input. The usual CLIQUE algorithm makes several passes over the database. Each pass iterates over the data objects and for each data object, the selectivity counters of corresponding grid units are incremented by one. When adapting to microcluster input, we could use passes over the microclusters instead of passes over the data objects. Since microclusters can span over several grid units, the contribution of one microcluster to the selectivity of each grid unit must be determined carefully. This consists of calculating the expected value of the number of microcluster members falling into each grid unit, based on the microcluster statistics. Compared to the resampling method, this alternative approach could potentially increase the speed and at the same time avoid noise introduced by random sampling from the multivariate normal distribution. On the other hand, the resampling method is significantly simpler to implement and it is also conceptually clearer, since it can be used with any offline algorithm, not only grid-based ones. Taking all of this into account, the evaluation will be performed on the simpler, resampling-based method.

CluStream and DenStream were not designed to work with high-dimensional data, which is different in CLIQUE. In particular, they do not consider subspaces of the input space. This can be problematic. As noted in the introduction, distance measures, and in particular, nearest neighbor lookups become meaningless in high-dimensions, therefore CluStream's search for the microcluster with the nearest centerpoint becomes essentially a random selection procedure. To address this and related issues, Aggarwal et al. proposed a newer algorithm called HPStream[2], which performes better in high-dimensional settings.

4 Evaluation

In this section, we will evaluate CLIQUE combined with microclustering algorithms on streaming data. The evaluation is done using the SubspaceMOA Framework by Hassani et al.[16]

4.1 Datasets

An evolving synthetic dataset and a real dataset are considered.

The synthetic dataset is 3-dimensional and contains 4000 data objects that form two clusters, and each cluster has two relevant dimensions. The clusters keep changing as the stream evolves.

The real dataset (network intrusion data from KDDCup 1999) suffers from some issues. It has many attributes that have the same value over long time in the stream causing the algorithms to break down due to matrix singularities. Even after removal of all non-numeric and quasi-non-numeric (constant over long time) attributes (26 removed attributes in total), the problem persists. Tavallaee et al. analyzed this dataset and suggested an improved version of it, called NSL-KDD[25]. By simply removing non-numeric attributes, this data still remains not processable in our context. However, after the removal of 26 attributes that are quasi-non-numeric, the dataset becomes possible to process. The removed attributes are 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29, 38. Since 15 attributes still remain in the dataset, the evaluation remains meaningful.

4.2 The evaluation measures

Subspace clustering needs careful evaluation techniques due to its special characteristics compared to normal clustering. Hassani et al. evaluated several such metrics[15]. The SubspaceMOA framework has the following metrics: 1.0-CE, CMM, Entropy, F1, Purity, 1.0-RNIA, Rand statistic and SubCMM.

All algorithms were used with their default settings in SubspaceMOA.

4.3 Results

As we can see in Tables 1-4, the combination of CluStream and CLIQUE performs slightly better than DenStream with CLIQUE. We must note, however, that the 3 dimensional evaluation dataset is not representative for high-dimensional data. (Also note that there is a problem involving the SubCMM measurement, causing the DenStream evaluations to have zero result.)

Table 1. Results of CluStream+CLIQUE on the synthetic dataset (measured after 1000, 2000, 3000 and 4000 data objects, followed by the mean values)

numObjects	**1.0-CE**	**CMM**	**Entropy**	**F1**	**Purity**	**1.0-RNI**	**Rand st**	**SubCMM**
1000	0.27141	0.91609	0.57573	0.72022	0.64185	0.32027	0.94426	0.92367
2000	0.33483	0.90376	0.58031	0.6462	0.64424	0.41674	0.94936	0.67437
3000	0.24376	0.52621	0.13979	0.84106	0.64	0.25208	0.5003	0.53663
4000	0.21241	0.51352	0.13769	0.82384	0.712	0.25316	0.5003	0.49951
mean values	0.2656	0.71489	0.35838	0.75783	0.65952	0.31056	0.72356	0.65854

Table 2. Results of DenStream+CLIQUE on the synthetic dataset (measured after 1000, 2000, 3000 and 4000 data objects, followed by the mean values)

numObjects	**1.0-CE**	**CMM**	**Entropy**	**F1**	**Purity**	**1.0-RNI**	**Rand st**	**SubCMM**
1000	0	0.91609	0.53429	0.6904	0.64082	0	0.94426	0
2000	0	0.90376	0.55132	0.3395	0.51699	0	0.94936	0
3000	0	0.15346	0.14201	0.69872	0.67387	0	0.5003	0
4000	0	0.36953	0.0332	0.67735	0.58155	0	0.5003	0
mean values	0	0.58571	0.3152	0.60149	0.60331	0	0.72356	0

Table 3. Results of CluStream+CLIQUE on the reduced NSL-KDD dataset

1.0-CE	**CMM**	**Entropy**	**F1**	**Purity**	**1.0-RNI**	**Rand st**	**SubCMM**
4.3E-4	0.86318	0.7008	0.66651	0.78043	0.00125	0.99635	0.92804

Table 4. Results of DenStream+CLIQUE on the reduced NSL-KDD dataset

1.0-CE	**CMM**	**Entropy**	**F1**	**Purity**	**1.0-RNI**	**Rand st**	**SubCMM**
0.0	0.89291	0.62465	0.66651	0.7168	0.0	0.99635	0.0

5 Conclusion

We have discussed in this seminar paper why high-dimensional and streaming data applications require specialized treatment. In a major part of the paper we have seen a detailed explanation of CLIQUE with pseudocodes that describe the algorithm more precisely. We have also examined two microclustering approaches that can be used in combination with CLIQUE to work on streaming data. Finally we have evaluated both combinations on a real and a synthetic dataset in SubspaceMOA. The evaluation yielded the result that CluStream outperforms DenStream for the used datasets.

In future, further analysis on larger-scale high-dimensional streaming data and comparisons with newer algorithms would be interesting topics to investigate.

References

1. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.
2. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 852–863. VLDB Endowment, 2004.
3. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998.
4. I. Assent, R. Krieger, E. Muller, and T. Seidl. INSCY: Indexing subspace clusters with in-process-removal of redundancy. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 719–724. IEEE, 2008.
5. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *Database Theory ICDT'99*, pages 217–235. Springer, 1999.
6. A. Bifet. Adaptive learning and mining for data streams and frequent patterns. *ACM SIGKDD Explorations Newsletter*, 11(1):55–56, 2009.
7. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 328–339, 2006.
8. Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.
9. A. Da Silva, R. Chiky, and G. Hebrail. Clusmaster: A clustering approach for sampling data streams in sensor networks. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 98–107. IEEE, 2010.
10. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.
11. M. M. Gaber. Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85, 2012.

12. M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.
13. S. Goil, H. Nagesh, and A. Choudhary. MAFIA: Efficient and scalable subspace clustering for very large data sets. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 443–452, 1999.
14. S. Günnemann, E. Müller, I. Färber, and T. Seidl. Detection of orthogonal concepts in subspaces of high dimensional data. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1317–1326. ACM, 2009.
15. M. Hassani, Y. Kim, S. Choi, and T. Seidl. Effective evaluation measures for subspace clustering of data streams. In *Trends and Applications in Knowledge Discovery and Data Mining*, pages 342–353. Springer, 2013.
16. M. Hassani, Y. Kim, and T. Seidl. Subspace MOA: Subspace stream clustering evaluation using the MOA framework. In *Database Systems for Advanced Applications*, pages 446–449. Springer, 2013.
17. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.
18. K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *Proc. SDM*, volume 4, 2004.
19. H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, Mar. 2009.
20. J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.
21. G. Moise and 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*, pages 533–541. ACM, 2008.
22. E. Muller, I. Assent, S. Gunnemann, R. Krieger, and T. Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 377–386. IEEE, 2009.
23. L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004.
24. K. Sequeira and M. Zaki. SCHISM: A new approach for interesting subspace mining. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 186–193. IEEE, 2004.
25. M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. A detailed analysis of the KDD cup 99 data set. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications 2009*, pages 1–6, July 2009.
26. Wikipedia. Clustering high-dimensional data — Wikipedia, the free encyclopedia, 2013. [Online; accessed 13-December-2013].