Subspace clustering for hierarchical fuzzy system construction

Conference Paper · February 2002

DOI: 10.1109/FUZZ.2002.1005043 · Source: IEEE Xplore

2 authors, including:

Tom Gedeon
Australian National University
430 PUBLICATIONS **5,943** CITATIONS

SEE PROFILE

Subspace Clustering For Hierarchical Fuzzy System Construction

Chong A., Gedeon T.D.
School of Information Technology
Murdoch University
South Street, Murdoch
6150 Western Australia

Abstract – Hierarchical fuzzy systems are proposed to deal with the rule explosion problem of traditional fuzzy systems. The inference operations of the fuzzy systems are well established. The next step is to tackle the problem of finding subspaces for automated hierarchical fuzzy system construction. In this paper, we propose a clustering technique designed specifically for this purpose. It is both theoretically and experimentally confirmed that the algorithm has reasonable accuracy and scalability.

I. INTRODUCTION

Fuzzy systems suffer from rule explosion. To model a system with k variables and maximum T fuzzy terms in each dimension, the number of necessary rules is T^k, which will be very large if k is not very small. Because of this, fuzzy systems are limited to handling only very few variables.

Hierarchical fuzzy systems are designed to tackle this problem [1]. The basic idea is closely related to the concept of a subspace (see section 3 for details). The inference operations of hierarchical fuzzy systems have been established in [1]. Hence, emphasis should now be placed on the automated construction of such hierarchical rule base from data. Clustering is one of the most important techniques for fuzzy system generation from sample input-output data.

Given a set of data, clustering technique partitions the data into several groups such that the degree of association is strong within one group and weak for data in different groups. Emerging hierarchical fuzzy rule extraction places some special requirements on the clustering technique.

II. CLUSTERING REQUIREMENTS

The clustering requirements specific for hierarchical fuzzy rule extraction are presented as follows.

1. *Capable of handling high-dimensional data:* The ultimate goal of a hierarchical fuzzy system is to break the limitation of fuzzy systems in the maximum number of variables that is manageable. If the goal is achieved, fuzzy systems may be used to handle data with large number of dimensions. Hence, the clustering technique designed for the construction of hierarchical fuzzy systems must be able to handle high dimensional data.

2. *Interpretability of clusters produced:* One of the advantages of fuzzy systems that distinguish it from neural networks is its ability to explain its inference results. Once a conclusion is reached, the user can observe the rules fired to gain insights on how and why the conclusion is reached. The interpretability of a fuzzy system relates directly to the fuzzy rules used. The clustering technique used to generate fuzzy rules should be designed to produce clusters that constitute to easy-to-interpret fuzzy rules.

3. *No prior knowledge about data required:* Often, clustering techniques requires certain input parameters from users. These techniques are usable in situation where the users possess prior information about the data being studied. One of the important goals of generic fuzzy systems modeling is to help users model a problem domain without requiring any prior knowledge about the domain. In this case, the clustering technique should not require prior knowledge about the data being studied from the user.

III. SUBSPACE CLUSTERING

Let

$$X = \underset{i \in I}{\mathrm{X}} x_i \qquad eqn\ 3.1$$

be the k dimensional data space, where I = {1, 2, …, k } is the set of dimension indexes. Then

$$S = \underset{i \in I_0}{\mathrm{X}} x_i \qquad eqn\ 3.2$$

is a subspace of the full space, where $I_0 \subset I$.

0-7803-7280-8/02/$10.00 ©2002 IEEE

A subspace cluster is defined as a cluster that is embedded in a certain subspace. Figure 3.1 shows two one-dimensional subspace clusters embedded in dimension X1 and X2 respectively. Cluster C1 can be identified by observing its projection on X1. The data points in cluster C1 are spread uniformly across X2.

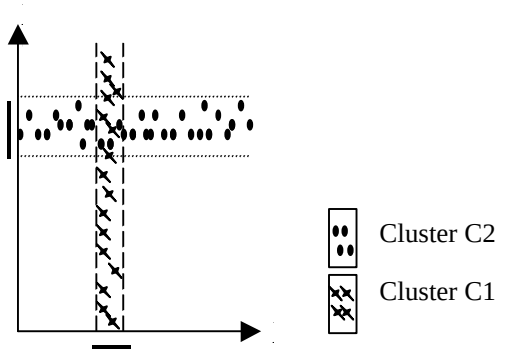

Figure 3.1 Subspace clusters C1 and C2 embedded in dimensions X1 and X2 respectively

The existence of subspace clusters in data introduces new problems for distance function based clustering algorithms. Consider a set of data with dimensions 1…*k*. If there exists a subspace cluster embedded in dimension 1…k_0 where k_0 is significantly smaller than *k*, then the data points in the cluster are distributed uniformly in dimension k_0…*k*. In this case, it becomes difficult for distance functions that use all dimensions of the data to reflect the associations among data points within the cluster.

According to our review of the clustering literature, CLIQUE [2] is one of the first clustering techniques designed specifically to find subspace clusters. ENCLUST [3] extends the idea of CLIQUE to exploit the concept of entropy. In this study, we modified CLIQUE to reduce the algorithm's computational complexity.

IV. CLIQUE

In this section, the algorithm of CLIQUE [2] is discussed. The basic idea of CLIQUE is as follows. The multi-dimensional data space is first partitioned into non-overlapping hyperboxes. This is done by partitioning every dimension into *e* number of equal-length intervals where *e* is an input parameter. Each hyperbox is the intersection of one interval from each dimension. A data point is said to be contained in a hyperbox if its projections on all dimensions are within the intervals that comprise the hyperbox. A hyperbox is dense if the number of points in it exceeds a threshold *t*, which is another user input. Similarly, a unit is defined to be the intersection of interval(s) from one or more dimensions.

Once all the dense units are found, clusters can be formed by connecting neighboring units. The core of the clustering technique lies in the algorithm to identify dense units. The algorithm is based on the Apriori algorithm [4] used extensively in data mining. The algorithm proceeds in multiple passes. In the first pass, all the one-dimensional units are examined and the dense units becomes candidates for the next pass. In general, having determined (k-1) dimensional dense units, the candidate k-dimensional units are determined using the candidate generation procedure given below.

C_k = set of candidates at pass *k*
$u.a_i$ = i^{th} dimension of unit *u*
$u.[l_i,h_i)$ = interval in the i^{th} dimension
D_{k-1} = set of all (k-1) dimensional dense units

insert into C_k
select $u_1.[l_1,h_1), u_1.[l_2,h_2), \ldots, u_1.[l_{k-1},h_{k-1}), u_2.[l_{k-1},h_{k-1})$
from D_{k-1} u_1 D_{k-1} u_2
where $u_1.a_1 = u_2.a_1, u_1.l_1 = u_2.l_1, u_1.h_1 = u_2.h_1,$
$u_1.a_2 = u_2.a_2, u_1.l_2 = u_2.l_2, u_1.h_2 = u_2.h_2, \ldots,$
$u_1.a_{k-2} = u_2.a_{k-2}, u_1.l_{k-2} = u_2.l_{k-2}, u_1.h_{k-2} = u_2.h_{k-2},$
$u_1.a_{k-1} = u_2.a_{k-1}, u_1.l_{k-1} = u_2.l_{k-1}, u_1.h_{k-1} = u_2.h_{k-1}$

The relation < represents lexicographic ordering on dimensions. Upon candidate generation, dense units that have a projection in (k-1)-dimensions that is not included in C_{k-1} are discarded. The resulting candidates then go through the Minimal Description Length (MDL-based) pruning stage.

Given the subspaces $s_1, s_2, \ldots, s_n$, the MDL-based technique first groups together the dense units that lie in the same subspace. Then for each subspace, the coverage is:

$$coverage\ (s_j) = \sum_{u_i \in s_j} count\ (u_i) \qquad eqn\ 4.1$$

Where count(u_i) is the number of points that is contained in u_i. Subspaces with small coverage are pruned.

The algorithm terminates when no more candidates are left for a particular pass. Using a bottom-up approach and discarding non-dense units in the early passes, the algorithm is able to prune a significant volume of the error space. The MDL-based pruning method further discards candidates that are less likely to be clusters, increasing the speed of the algorithm. We remark that the MDL-based pruning method can be error prone. Figure 4.1 shows situation where MDL-based pruning can be ineffective. In the figure, the bolded units are more likely to be retained than those real cluster units due to their high coverage.

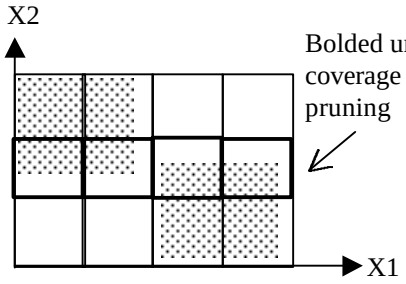

Figure 4.1 Coverage of units

Even with the pruning strategies introduced in [2], the algorithm still suffers from high computational complexity. This is explained as follows. If a dense unit exists in *k*-dimensions, then all of its projections in a subset of *k*-dimensions is also dense. The total number of combinations to be explored by the algorithm to identify the dense unit is calculated as

$$\sum_{i=1}^{k} \binom{k}{i} \quad \text{where} \quad \binom{k}{i} = \frac{k!}{k\,(k-i)!} \qquad \text{eqn } 4.2$$

The overall complexity of the algorithm is thus, $O(c^k)$ for some constant *c*. Therefore, improvement on the algorithm to reduce the computational complexity is necessary. In the next section, we present our modified algorithm with reduced complexity.

V. MODIFIED CLIQUE

One of the goals of the proposed clustering technique is to produce clusters for the construction of hierarchical fuzzy system. Since fuzzy rules operate on the projections of the multi-dimensional clusters, convex clusters are desired for the fuzzy system generation. Hence, one of the main differences between our algorithm and the original CLIQUE is that our algorithm is designed to approximate convex clusters.

The basic idea of CLIQUE is retained in our modified algorithm. The algorithm starts by partitioning every dimension into *e* (input parameter) number of equal-length intervals. A unit is considered dense if the number of data points contained in the unit exceeds the threshold *t* (input parameter). The 4-step algorithm as well as the pseudo code for the important procedures involved is presented.

1. Find one of the dense units, *u*, that exceeds the threshold *t*.
2. Approximate convex cluster, *C*, by expanding the dense unit *u* in each of the dimensions that the dense unit is embedded in.
3. Remove all data points that are contained in the cluster *C* approximated.
4. Repeat steps 1 – 3 until no dense unit can be found.

```
PROCEDURE find_dense_unit

Let U_i be the set of one-dimensional units in dimension i
Let denseunit = [ ]
for i = 1 to k
   for each unit u ∈ U_i
      utemp = denseunit x u
      if utemp is dense
        denseunit = utemp
        break
      end if
   end for
end for
```

For the convenience of discussion, we define [ ] as the zero-dimensional (empty) subspace where [ ] x X_i = X_i. The procedure scans through each of the dimensions to find one of the dense units in the data.

```
PROCEDURE approximate_convex_cluster(u)

Let D be the set of dimension indexes of u
for each i in D
  Let clusterset = {}
  expand_along(u,i)

Let ul = left most element of clusterset along dimension i
Let ur = right most element of clusterset along
           dimension i
  u.l = ul.l
  u.h = ur.h
end for
```

Given a dense unit, this procedure expands the unit along all the dimensions that the unit is embedded in. The procedure results in a hyper-rectangular cluster.

```
PROCEDURE expand_along(u,i)

global clusterset
add u to clusterset

Let u^r = right neighboring unit of u along dimension i
  if u^r is dense
    expand_along(u^r,i)
  end if

Let u^l = left neighboring unit of u along dimension i
if u^l is dense
   expand_along(u^l,i)
end if
```

This procedure is used by approximate_convex_cluster to expand a dense unit along a certain dimension. Using the modified algorithm, the computational complexity is greatly reduced. To find a dense unit that exists in *k*-dimensions, the procedure find_dense_unit performs a single pass scan through each dimension of the data, giving the complexity O(k). To approximate a convex cluster using the *k*-dimensional dense unit, the procedure approximate_convex_cluster examines each of the *k* dimensions O(*k*) by calling the procedure expand_along. The procedure expand_along examines both the right and left neighboring units. In the worse case, all *e* number of units are examined O(*e*). The algorithm terminates when all clusters are found (by then, all data points would have been removed). Thus the overall complexity is:

$$O(c \times (k + ke)) = O(cke)$$

Since the complexity of the algorithm is linear, it is computational feasible to cluster data with very large number of dimensions.

VI. EXPERIMENTAL RESULTS

In this section, the performance of the proposed algorithm is evaluated. The overall goal of the experiments is to evaluate the efficiency and accuracy of the algorithm. In terms of efficiency, the experiments aim to verify the scalability of the algorithms in:

1. The dimensionality of the data space
2. The dimensionality of clusters
3. Number of data points

The synthetic data generator from [5] is used to produce the data for the experiments. Figure 6.1 shows a sample input to the data generator to generate two-dimensional data. In the figure, clusters 2 and 3 are subspace clusters embedded in dimensions X2 and X1 respectively. A total of 3300 data points are generated in which 10% of the data is noise.

Cluster	X1	X2	Number of Points
1	[0.2, 0.3]	[0.2, 0.3)	10000
2	[0.0, 1.0]	[0.5, 0.6)	10000
3	[0.8, 0.9)	[0.0, 1.0)	10000
Noise	[0.0, 1.0)	[0.0, 1.0)	3000

Figure 6.1 Sample input to data generator

The clusters generated are hyper-rectangles in shape and data points are uniformly distributed within the cluster. The rest of this section discusses the results of the experiments. Figure 6.2b, 6.3b, and 6.4b are extracted from [2] to allow for comparisons.

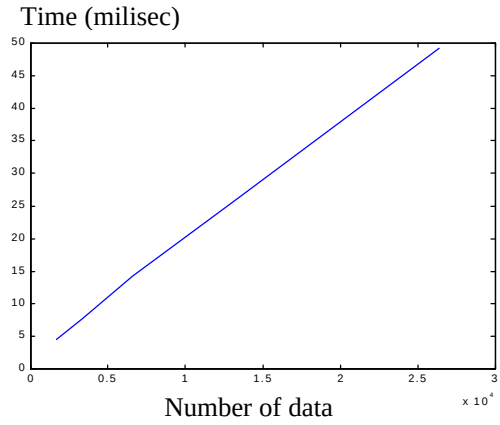

Figure 6.2a Scalability with the number of data

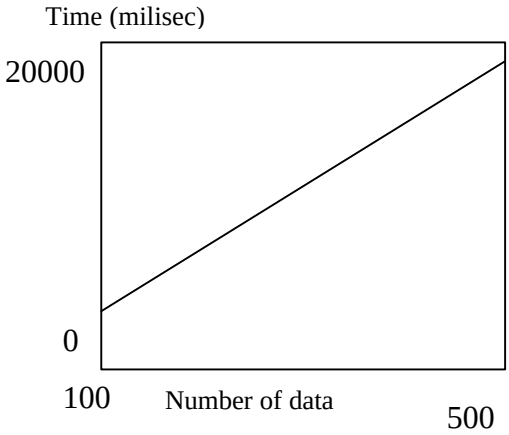

Figure 6.2b CLIQUE scalability with the number of data

Figure 6.2a shows the scalability of the algorithm as the number of data increases. The data has five dimensions. The threshold *t* is chosen as 0.3. There are three clusters embedded in the full space. The number of data points increases from 1650 to 26400. From the figure, our algorithm scales linearly with the increase of the number of data. It is shown in figure 6.2b that CLIQUE performs equally well in terms of its scalability.

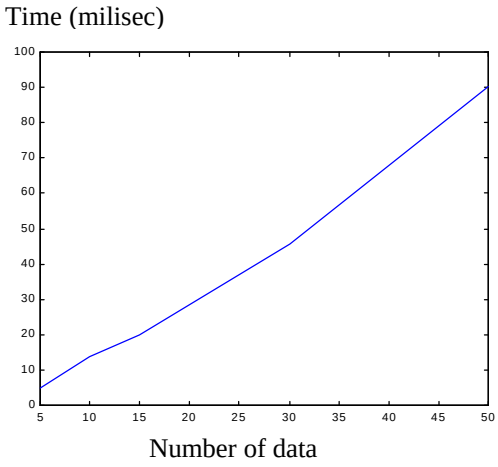

Figure 6.3a Scalability with the data space dimensionality

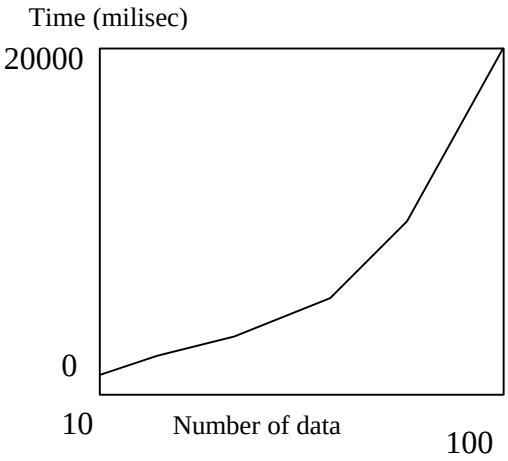

Figure 6.3b CLIQUE scalability with the data space dimensionality

Figure 6.3a shows the scalability of the algorithm as the number of dimensions of the data increases. There are three five dimensional subspace clusters. The threshold t = 0.3 is used. There is 26400 data points. The number of dimensions of the data space increases from 5 to 50 dimensions. Again, our algorithm scales linearly with the increase of the data space dimensionality. Figure 6.3b shows a quadratic behavior for CLIQUE.

Figure 6.4a shows the scalability of the algorithm as the highest dimensionality of the hidden clusters increases. To keep the experiment simple, there is only one cluster in the data. The number of dimensions increases from 4 to 10 dimensions. Our algorithm scales linearly with the increase of cluster dimensionality. The performance of CLIQUE is shown in figure 6.4b.

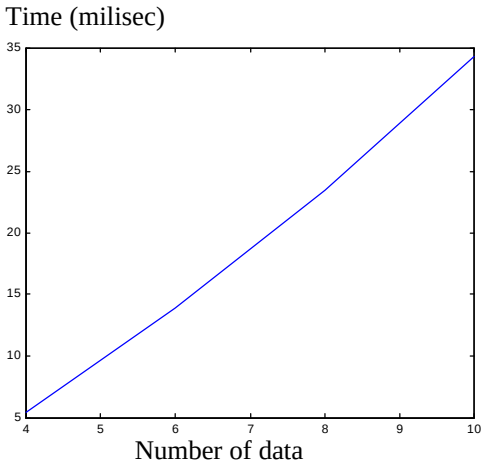

Figure 6.4a Scalability with the cluster dimensionality

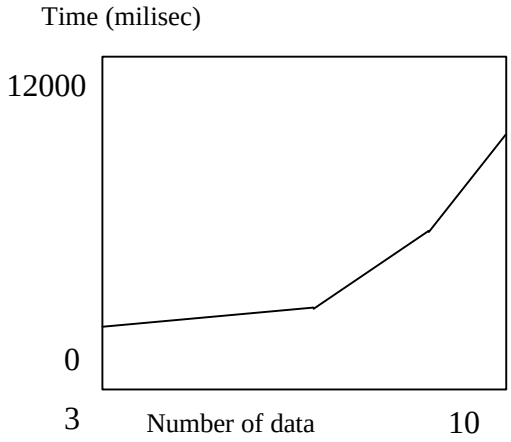

Figure 6.4b CLIQUE scalability with the cluster dimensionality

In all the above experiments, our algorithm is able to recover the original clusters in the data. Apart from that, it is easily observed that the algorithm scales linearly with the increase of the dimensionality of the data space, the dimensionality of clusters as well as the number of data. Overall, the experiments show that our algorithm outperforms CLIQUE significantly in every case.

VII. CONCLUSION

In this paper, we have proposed a subspace-clustering algorithm. The algorithm is designed to find convex subspace cluster that can be used for the construction of hierarchical fuzzy systems. Our algorithm is an improvement over CLIQUE, one of the first clustering technique designed to find subspace cluster. It is both theoretically and experimentally confirmed that the complexity of our algorithm is significantly reduced.

Since the computational complexity of our algorithm is low, it can be used to deal with high dimensional data. The clustering technique brings us one step nearer to the construction of hierarchical fuzzy systems. In our subsequent work, we will further extend the proposed algorithm to reduce the necessary user input parameters.

REFERENCES

[1] Koczy, L.T. *Approximative inference in hierarchical structured rule bases*. in *Fift IFSA World Congress*. 1993. Seoul: International Fuzzy Systems Association.

[2] Agrawal, R., Gehrke, J., Gunopulon, D., and Raghawan, P. *Automatic subspace clustering of high dimensional data for data mining applications*. in *Proceedings of the ACM SIGMOD conference on Management of Data*. 1998. Canada.

[3] Cheng, C.H., Fu, A.W., and Zhang, Y. *Entropy-based Subspace Clustering for Mining Numerical Data*. in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)*. 1999. San Diego.

[4] Agrawal, R. and Srikant, R. *Fast algorithms for mining association rules*. in *Proceedings of the 20th VLDB Conference*. 1994.

[5] Zait, M. and Messatfa, H., *A Comparative Study of Clustering Methods*. Future Generation Computer Systems, 1997. **13**: p. 149-159.

0-7803-7280-8/02/$10.00 ©2002 IEEE