Frequent-Pattern based Iterative Projected Clustering [*]

Man Lung Yiu and Nikos Mamoulis
Department of Computer Science and Information Systems
University of Hong Kong
Pokfulam Road, Hong Kong
{mlyiu2,nikos}@csis.hku.hk

Abstract

Irrelevant attributes add noise to high dimensional clusters and make traditional clustering techniques inappropriate. Projected clustering algorithms have been proposed to find the clusters in hidden subspaces. We realize the analogy between mining frequent itemsets and discovering the relevant subspace for a given cluster. We propose a methodology for finding projected clusters by mining frequent itemsets and present heuristics that improve its quality. Our techniques are evaluated with synthetic and real data; they are scalable and discover projected clusters accurately.

1. Introduction

Clustering is typically used to partition a collection of data samples into a set of clusters (i.e., groups) such that the similarity between objects within a cluster is large and the objects from different clusters are dissimilar. Typical applications include customer segmentation, image processing, biology, document classification, indexing, etc.

It was shown in [5] that the distance of any two records is almost the same in high dimensional spaces for a large class of common distributions. Thus, the widely used distance measures are more meaningful in subsets (i.e., *projections*) of the high dimensional space. It is more likely for the data to form dense, meaningful clusters in a dimensional subspace, especially in real datasets, where irrelevant, noise attributes exist. The effects of dimensionality can be reduced by a dimensionality reduction technique [7] but information from all dimensions is uniformly transformed and relevant information for some clusters may be reduced. Also, clusters in the transformed space may be hard to interpret.

Therefore, *projected clustering* methods have been developed to find the clusters together with their associated subspaces. These methods disregard the noise induced by irrelevant dimensions and also provide interpretable descriptions for the clusters. CLIQUE [3], one of the first projected clustering algorithms, finds the dense regions (clusters) in a level-wise manner, based on the Apriori principle. However, this algorithm does not scale well with data dimensionality. In addition, the formed clusters have large overlap, and this may not be acceptable for some applications (e.g., classification) which require disjoint partitions.

PROCLUS [1] and ORCLUS [2] employ alternative techniques. They are much faster than CLIQUE and they can discover disjoint clusters. In PROCLUS, the dimensions relevant to each cluster are selected from the original set of attributes. ORCLUS is more general and can select relevant attributes from the set of arbitrary directed orthogonal vectors. PROCLUS fails to identify clusters with large difference in size and requires their dimensionality to be in a predefined range. ORCLUS may discover clusters that are hard to interpret.

DOC [10] is a simple, density-based, projected clustering algorithm. A projected cluster C is defined by (i) the set of points in C (also denoted by C), (ii) a set D of relevant dimensions. In addition, three parameters w,α, and β are defined. w controls the extent of the clusters; the distances between records in the same cluster in each relevant dimension are bounded by w. $\alpha \in (0, 1]$ is the minimum *density* of the discovered clusters; each cluster should have at least $\alpha \cdot |S|$ points, where $|S|$ is the database size. $\beta \in (0, 1]$ reflects the importance of the size of the subspace over the size of the cluster. DOC discovers one cluster at a time. At each step, it picks a random point p from the database S and attempts to discover the cluster centered at p. For this, it runs an inner loop that selects a set of samples $X \subset S$. A set of dimensions D, where *all* points in X are within distance w from p is selected. Then, a cluster C for X is approximated by a bounding box of width $2w$ around p in the relevant dimensions. C is defined by the set of points from S in this box. The process is repeated for a number of random points p and samples X for each p. Among all discovered

*This research was supported by grant HKU 7380/02E from Hong Kong RGC.

C, the cluster with the highest *quality* is finally selected. The quality of a cluster C is defined by $\mu(a,b) = a \cdot (1/\beta)^b$, where a is the number of points in C and b is the dimensionality of C. After one cluster has been discovered, the records in it are removed from the sample and the process is iteratively applied to the rest of the points. With this approach the number of clusters k can be automatically found. Moreover, small clusters can be identified. However, DOC only produces approximate results and requires a lot of time to discover clusters of high accuracy.

In this paper, we propose an algorithm that improves DOC in several ways. First, we draw some analogues between mining frequent itemsets and discovering the relevant subspace for a given cluster. Then, we adapt a data mining technique [9] to systematically find the optimal cluster. Second, we propose techniques that improve the quality of the clusters. The resulting algorithm is much faster than DOC, while producing clusters of high quality.

The outline of the paper is as follows. Our methodology is presented in Section 2. Section 3 presents experimental comparisons between projected clustering techniques. Finally, Section 4 concludes the paper and discusses issues for future work.

2. Projected Clustering

2.1. Mining relevant dimensions

Given a random medoid $p \in S$, we can transform the problem of finding the best projected cluster that contains p, to the problem of mining frequent itemsets in transactional databases as shown in Figure 1. The original dataset S is shown in Figure 1a. We consider each dimension i as one attribute a_i. Assume that the record marked in bold is the medoid p. We replace each point $q \in S$ by an itemset as follows. If and only if the value of q in dimension i is bounded by p with respect to width w (here, w=2), we include a_i in the corresponding itemset, as shown in Figure 1b. Observe that all frequent itemsets (i.e., combinations of dimensions) with respect to $min\_sup = \alpha \cdot |S|$ are candidate clusters for medoid p.

a_1	a_2	a_3	a_4
1	2	4	8
2	1	9	6
3	**2**	**7**	**3**
4	8	1	2
9	6	2	1
7	3	3	2

(a) Original table

Itemset
$\{a_1, a_2\}$
$\{a_1, a_2\}$
$\{a_1, a_2, a_3, a_4\}$
$\{a_1, a_4\}$
$\{a_4\}$
$\{a_2, a_4\}$

(b) Corresponding itemsets

Figure 1. Transforming dataset to itemsets

```
/* Table header entries (hl) are in descending order of support */
Algorithm $\mu Growth(\mathcal{T}, \mathcal{I}_{cond}, \mathcal{I}_{best})$
1   if $\mathcal{T}$ has a single path then
2       $\mathcal{I} := \mathcal{I}_{cond}$;
3       for $l$:=1 to $\mathcal{T}.hl.length$
4           $\mathcal{I} := \mathcal{I} \cup \{\mathcal{T}.hl[l].item\}$; $sup(\mathcal{I}) := \mathcal{T}.hl[l].support$;
5           update $\mathcal{I}_{best}$ if $\mu(\mathcal{I}) > \mu(\mathcal{I}_{best})$;
6   else
7       $\mathcal{I} := \mathcal{I}_{cond} \cup \{\mathcal{T}.hl[\mathbf{1}].item\}$; $sup(\mathcal{I}) := \mathcal{T}.hl[\mathbf{1}].support$;
8       update $\mathcal{I}_{best}$ if $\mu(\mathcal{I}) > \mu(\mathcal{I}_{best})$;
9       for $l$:=$\mathcal{T}.hl.length$ down to 2
10          $\mathcal{I} := \mathcal{I}_{cond} \cup \{\mathcal{T}.hl[l].item\}$;
11          if $\mu(\mathcal{T}.hl[l].support, dim(\mathcal{I}_{cond}) + l) > \mu(\mathcal{I}_{best})$ then
12              construct $\mathcal{I}$'s conditional pattern base;
13              create $\mathcal{I}$'s conditional FP-Tree $\mathcal{T}_{\mathcal{I}}$;
14              if $(\mathcal{T}_{\mathcal{I}} \neq \emptyset)$ then $\mu Growth(\mathcal{T}_{\mathcal{I}}, \mathcal{I})$;
```

Figure 2. The $\mu Growth$ algorithm

Therefore, the problem of finding the best projected cluster for a random medoid p can be transformed to the problem of finding the best itemset in a transformation of S (like the one of Figure 1b), where goodness is defined by the μ function. Instead of discovering it in an non-deterministic way [10], we apply a systematic data mining algorithm on S. The frequent itemset mining problem was first proposed in [4]. Recently, there is an more efficient algorithm, the FP-growth method [9]. We adopt it for subspace clustering. However, our objective is to find the frequent itemset with maximum μ value, rather than finding *all* frequent subspaces with respect to p.

Assume that $\mathcal{I}_{best}$ is the itemset with the maximum μ value found so far and let $dim(\mathcal{I}_{best})$ and $sup(\mathcal{I}_{best})$ be its dimensionality and support, respectively. Let $\mathcal{I}_{cond}$ be the current conditional pattern of the FP-growth process. Its support $sup(\mathcal{I}_{cond})$ gives an upper bound for the supports of all patterns containing it. Moreover, the dimensionality of the itemsets that contain $\mathcal{I}_{cond}$ is at most $dim(\mathcal{I}_{cond}) + l$, where l is the number of items above the items in $\mathcal{I}_{cond}$ in the header table. Therefore if:

$$\mu(sup(\mathcal{I}_{cond}), dim(\mathcal{I}_{cond}) + l) \leq \mu(\mathcal{I}_{best}), \qquad (1)$$

we can avoid constructing the conditional FP-tree for $\mathcal{I}_{cond}$, since that tree cannot generate a better pattern than $\mathcal{I}_{best}$. This bound can help prune the search space of the original mining process, effectively.

The $\mu Growth$ process is shown in Figure 2. It can replace the randomized inner loop of DOC to systematically discover the best subspace for a given medoid p. Moreover, it can accelerate a given phase of DOC. The best μ found so far is kept, allowing further pruning in subsequent iterations. In other words, if a good p is found in early iterations, it can help prune FP-trees for other medoids in subsequent iterations. With this modification, DOC can converge to a good solution fast.

2.2. The MineClus algorithm

Our clustering algorithm (*MineClus*) has four phases. In the *iterative* phase, the process described in Section 2.1 is applied to generate iteratively one cluster at a time. It is possible that the resulting cluster may be part of a large cluster that spreads outside the bounding rectangle. By using the Manhattan segmental distance [1], we also assign records having distance at most max_dist from the cluster centroid. max_dist is defined by the distance of the farthest point from the centroid, currently in the cluster. In the *pruning* phase, clusters having μ values significantly lower than the rest are pruned. First, we sort the clusters according to their μ values in descending order. Then, we find the position pos such that $\mu_{pos}/\mu_{pos+1} \geq \mu_i/\mu_{i+1} \; \forall i$. This position divides the clusters into a set of *strong* clusters C_i $(i \leq pos)$, and a set of *weak* clusters C_i $(i > pos)$. The weak clusters are pruned and their records are added back to S. The *merging* phase is applied only when the user wants at most k clusters in the result. In this case, the strong clusters are merged in an agglomerative way until k clusters remain. Given clusters C_x and C_y, the merged cluster is $C_x \cup C_y$, its subspace is $D_x \cap D_y$, its spread[1] is $R(C_x \cup C_y, D_x \cap D_y)$ and its μ value is $\mu(|C_x \cup C_y|, |D_x \cap D_y|)$. A good cluster should have small spread and large μ value (i.e., large subspace), so we use both measures to determine the next pair to merge. We consider two rankings of the cluster pairs; one with respect to spread and one with respect to μ value. Then the pair with the highest sum of ranks in both orderings is merged. In the *refinement* phase, we further improve the clusters by assigning the remaining records in the dataset (considered as outliers so far) to clusters. We use a similar algorithm to the refinement phase of PROCLUS [1].

3. Experimental Evaluation

In this section, we experimentally evaluate the effectiveness and efficiency of MineClus by comparing it with DOC[2] and PROCLUS, under various settings, for synthetic and real data. The performance measures are accuracy, percentage of outliers, and running time. Clustering accuracy corresponds to the number of correctly classified samples as a percentage of the total number of clustered data (excluding outliers). Outlier percentage is defined by the number of records assigned to no clusters as a percentage of the database size. First, we compare the performance of the methods on synthetic data and study their scalability on large datasets. Then, we compare them on real datasets.

[1]the *spread* $R(C, D)$ of a cluster C is defined by the mean squared distance between its points and its centroid, considering only the relevant dimensions D [2].

[2]We also implemented FastDOC [10], a faster variant of DOC, but found that the clusters generated failed to satisfy the α constraint most of the cases and it was sensitive to outliers.

We have implemented a synthetic data generator similar to the one in [1]. The outlier percentage is 5%. The generated datasets contain $k = 5$ clusters with random subspaces comprising from 5 to 10 dimensions. The smallest cluster size is $0.1 \cdot |S|$, where $|S|$ is the size of the database S. In the experiments that involve synthetic data, the results are averaged over 5 runs in order to smoothen the effects of randomness in the algorithms. All algorithms were implemented in Java. The experiments were run on a PC with a Pentium 4 CPU of 2.3GHz and 512MB RAM.

First, we compare the accuracy of MineClus, DOC, and PROCLUS, for various types of synthetic data. The input parameters for MineClus and DOC are $\alpha = 0.08, \beta = 0.25, w = 0.2$. For PROCLUS, we set $k = 5$ and the average subspace dimensionality $l = 7$. The running time of DOC is too high when the number of inner iterations m is high. We set $m = 2^{10}$ for DOC because it has quite high accuracy with this value, and its running time is in the same order as MineClus and PROCLUS.

Figure 3 shows accuracy as a function of α and β, on the same synthetic dataset. Entries in the tables are of the form X/Y, where X is the accuracy percentage and Y is the outlier percentage. In general, both MineClus and DOC have high accuracy. When $\alpha = 0.12$, the accuracy of MineClus decreases as the smallest cluster with $0.1 \cdot |S|$ records was missed. MineClus is not sensitive to β because of the deterministic behavior of the $\mu Growth$ algorithm. The accuracy of DOC decreases significantly as β increases, because DOC picks a larger discriminating set and smaller subspaces are likely to be discovered. Observe that DOC misclassifies many points as outliers in both experiments. Figure 4 compares the accuracy of all three algorithms as a function of data dimensionality. Observe that the accuracy of MineClus (and PROCLUS) is insensitive to dimensionality. On the other hand, the accuracy of DOC decreases when the dimensionality increases. This is explained by the fact that DOC applies a fixed number m of inner iterations and the chance to select an appropriate sample in each iteration decreases with dimensionality.

Next, we compare the scalability of the algorithms on various dataset sizes. Figure 5 shows their running time in seconds. They are all scalable to the database size. However, DOC is very expensive compared to the other methods, even for the smallest value of m, where its accuracy is low. MineClus is the fastest technique due to the efficiency of the $\mu Growth$ algorithm.

Finally, we compare the effectiveness of the three algorithms on real datasets from the UCI Machine Learning Repository [6]. The Iris dataset has only numerical attributes and the rest have only categorical attributes so we set $w = 0.5$ for Iris dataset and $w = 0$ for the rest in both DOC and MineClus. These datasets have no outliers so we turned off the outlier removal mechanism. The number of clusters k is set to the number of classes, except from the mushroom dataset where k is set to 20 (as suggested in [8]). For PROCLUS, we set the average subspace dimensionality l to be the average subspace size of the actual clusters. For DOC and MineClus, we set $\beta = 0.25$ and $\alpha = 0.2, 0.2, 0.4, 0.01$ for the Iris, Soybean, Votes, and Mushroom datasets respectively. DOC becomes too slow for the Mushroom dataset, so no result is given. In general, MineClus and DOC have high accuracy but DOC declares too many points as outliers. In summary, MineClus is highly accurate and robust.

α	MineClus	DOC,$m = 2^{10}$
0.04	95.57/0.60	99.05/16.00
0.06	95.76/0.80	100.00/15.04
0.08	95.12/1.60	99.72/26.40
0.10	95.38/0.40	99.54/12.40
0.12	89.88/1.2	99.53/15.00

(a) Dependency on α

β	MineClus	DOC,$m = 2^{10}$
0.1	95.84/0.87	100.00/50.80
0.2	96.04/1.08	100.00/24.80
0.3	94.53/3.32	88.70/13.00
0.4	95.68/0.72	50.60/6.00

(b) Dependency on β

Figure 3. Accuracy and outlier percentage

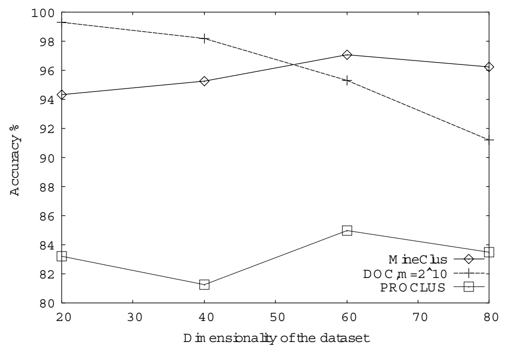

Figure 4. Accuracy w.r.t. dimensionality

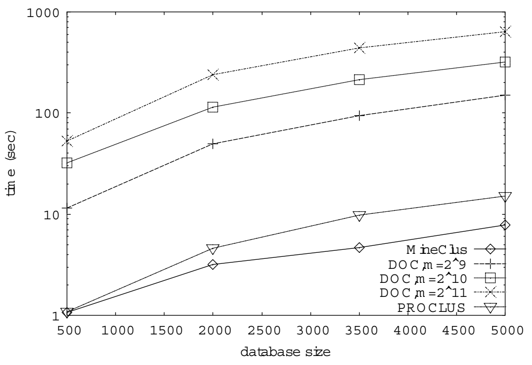

Figure 5. Scalability w.r.t. database size

4. Conclusions

In this paper, we presented an efficient and effective projected clustering algorithm. First, we identifed the similarity between mining frequent itemsets and discovering the best projected cluster for a pivot point p. Then, we proposed an adaptation of FP-growth that exploits the properties of the μ function and employs branch-and-bound techniques to reduce the search space significantly. We extended the cluster definition of [10] to consider more appropriate distance and quality measures for projected clustering. The quality of the results was further improved by (i) pruning small clusters of low quality, (ii) merging clusters close to each other with similar subspaces, and (iii) assigning points close to some cluster, else considered as outliers. We evaluated the efficiency and effectiveness of MineClus by comparing it with DOC [10] and PROCLUS [1] using synthetic and real data, under various conditions. It was shown that MineClus is more efficient, robust and scalable. In the future, we hope to devise additional heuristics for improving the discovered clusters.

Dataset	MineClus	DOC,$m = 2^{10}$	PROCLUS
Iris	92.66/0.00	94.80/22.00	87.87/0.00
Soybean	97.87/0.00	81.30/8.50	84.97/0.00
Votes	86.67/0.00	99.60/28.00	84.41/0.00
Mushroom	96.41/0.59	-/-	97.68/0.00

Figure 6. Performance on real datasets

References

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. In *ACM SIGMOD*, 1999.

[2] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *ACM SIGMOD*, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD*, 1998.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[5] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *ICDT*, 1999.

[6] C. Blake and C. Merz. UCI repository of machine learning databases, www.ics.uci.edu/∼mlearn/mlrepository.html, 1998.

[7] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD*, 1995.

[8] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *IEEE ICDE*, 1999.

[9] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD*, 2000.

[10] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *ACM SIGMOD*, 2002.