Scalable Algorithms for Clustering Large Datasets with Mixed Type Attributes

Zengyou He,* Xiaofei Xu,† Shengchun Deng‡
Department of Computer Science and Engineering,
Harbin Institute of Technology, Harbin 150001, P. R. China

Clustering is a widely used technique in data mining applications for discovering patterns in underlying data. Most traditional clustering algorithms are limited to handling datasets that contain either numeric or categorical attributes. However, datasets with mixed types of attributes are common in real life data mining applications. In this article, we present two algorithms that extend the Squeezer algorithm to domains with mixed numeric and categorical attributes. The performance of the two algorithms has been studied on real and artificially generated datasets. Comparisons with other clustering algorithms illustrate the superiority of our approaches. © 2005 Wiley Periodicals, Inc.

1. INTRODUCTION

Clustering typically groups data into sets in such a way that the intracluster similarity is maximized while the intercluster similarity is minimized. The clustering technique has been extensively studied in many fields such as pattern recognition, customer segmentation, similarity search, and trend analysis. The capability to deal with datasets that contain both numeric and categorical attributes is undoubtedly important due to the fact that datasets with mixed types of attributes are common in real life data mining applications. Although many clustering algorithms have been proposed so far, most of those algorithms are designed to find clusters on an assumption that all the attributes are either numeric or categorical.

In this article, we present two algorithms that extend the Squeezer algorithm in Ref. 1 to domains with mixed numeric and categorical attributes. This work is motivated by the following observations.

First, most existing clustering algorithms are designed to handle datasets that contain either numeric or categorical attributes. However, datasets with mixed types of attributes are common in real life clustering applications.
Second, previous proposed clustering algorithms for datasets with mixed types of attributes contain certain drawbacks in their own ways, which limits their spread usage and acceptance in many real world situations.

Finally, the success of the Squeezer algorithm in producing high-quality clustering results in high-dimensional categorical datasets motivates us to extend it to domains with mixed numeric and categorical attributes.

Based on the above observations, we present two algorithms: dSqueezer (Discretizing before using Squeezer) and usmSqueezer (Unified Similarity Measure based Squeezer). Since the Squeezer algorithm is effective for categorical datasets, in the dSqueezer algorithm we adopt the simple strategy of transforming the original dataset into categorical dataset with the discretization of numeric attributes. Then, the Squeezer algorithm is employed to cluster the transformed dataset. For the usmSqueezer algorithm, we define a unified similarity measure for mixed-type attributes, in which both numeric and categorical attributes could be handled equally in the framework of Squeezer algorithm.

The remainder of this article is organized as follows. Section 2 discusses related work. Section 3 gives a brief introduction on the Squeezer algorithm. In Section 4, the dSqueezer algorithm and usmSqueezer algorithm are proposed. Experimental results are given in Section 5, and Section 6 concludes the article.

2. RELATED WORK

From the viewpoint of target dataset for cluster analysis, existing clustering algorithms can be classified into three categories: numeric, categorical, and mixed. Most previous clustering algorithms focus on numerical data whose inherent geometric properties can be exploited naturally to define distance functions between data points, such as C2P,2 DBSCAN,3 BIRCH,4 CURE,5 CHAMELEON,6 and WaveCluster.7 As pointed out in Ref. 8, numeric clustering algorithms are not appropriate for categorical attributes; therefore, it is easy to conclude that they are also not suitable for the task of clustering mixed-type attributes.

Recently, some effective and efficient algorithms have been proposed for clustering categorical data.1,8–14 However, all of these algorithms are designed for categorical attributes and their potential capability on clustering mixed type attributes are unknown.

For the problem of clustering mixed type attributes, some research efforts have been done.15–17 In Ref. 15, Huang presents two algorithms, k-modes and k-prototypes, which extend the k-means paradigm to categorical domains and domains with mixed attributes. A new distance measure for categorical attributes based on the total mismatches of the categorical attributes of two data records is proposed in the k-modes algorithm. For the mixed type of attributes, the k-prototypes algorithm used a weighted sum of Euclidean distance. However, the weights have to be determined a priori. Improper weights may result in biased treatment of different attributes types. Furthermore, the k-prototypes algorithm needs multiple scans over the whole dataset, which is not appropriate in large-scale data mining applications.
In Ref. 16, the SBAC algorithm is proposed, which adopts a similarity measure that gives greater weight to uncommon attribute value matches in similarity computations and employs an agglomerative algorithm to construct a dendrogram. However, the complexity of the SBAC algorithm is quadratic in the number of records of the dataset; it is almost impossible to handle very large datasets.

In Ref. 17, Chiu et al. introduce their clustering algorithm that is available commercially in the Clementine 6.0 data mining tool. Their distance measure is derived from a probabilistic model that the distance between two clusters is equivalent to the decrease in log-likelihood function as a result of merging. Their clustering algorithm is based on the framework of BIRCH. The BIRCH algorithm has the drawback that it may not work well when clusters are not “spherical” and could be affected by the input order of records. Thus, Chiu et al.’s algorithm has the same limitation.

In summary, limitations of earlier methodologies prompted us to look for scalable clustering algorithms that would give better clustering results for the dataset with mixed type of attributes.

3. SQUEEZER ALGORITHM

Our new clustering algorithms are based on the Squeezer algorithm. Hence, before proceeding any further, we will first give an introduction on Squeezer algorithm.

Let $A_1,\ldots,A_m$ be a set of categorical attributes with domains $D_1,\ldots,D_m$, respectively. Let the dataset $D$ be a set of tuples where each tuple $t: t \in D_1 \times \cdots \times D_m$. Let $TID$ be the set of unique ID of every tuple. For each $tid \in TID$, the attribute value for $A_i$ of corresponding tuple is represented as $a_{\text{tid} \cdot A_i}$.

**Definition 1 (Cluster).** $\text{Cluster} = \{\text{tid} \mid \text{tid} \in TID\}$ is subset of $TID$.

**Definition 2.** Given a Cluster $C$, the set of attribute values on $A_i$ with respect to $C$ is defined as $\text{VAL}_i(C) = \{a_{\text{tid} \cdot A_i} \mid \text{tid} \in C\}$.

**Definition 3.** Given a Cluster $C$, let $a_i \in D_i$, the support of $a_i$ in $C$ with respect to $A_i$ is defined as $\text{Sup}(a_i) = |\{\text{tid} \mid \text{tid} \cdot A_i = a_i\}|$.

**Definition 4 (Summary).** Given a Cluster $C$, the Summary for $C$ is defined as $\text{Summary} = \{\text{VS}_i | 1 \leq i \leq m\}$ where $\text{VS}_i = \{(a_j, \text{Sup}(a_j)) | a_j \in \text{VAL}_i(C)\}$

Intuitively, the Summary of a Cluster contains summary information about this Cluster. In general, each Summary consists of $m$ elements, where $m$ is the total number of attributes. Each element in Summary is the set of pairs of attribute values and their corresponding supports. We will show later that information contained in Summary is sufficient enough to compute the similarity between a tuple and Cluster.
Definition 5 (Cluster Structure, CS). Given a Cluster C, the Cluster Structure (CS) for C is defined as: $CS = \{\text{Cluster, Summary}\}$.

Definition 6. Given a Cluster C and a tuple t with tid $\in$ TID, the similarity between C and tid is defined as

$$Sim(C, tid) = \sum_{i=1}^{m} \left( \frac{Sup(a_i)}{|C|} \right) \text{ where } a_i = tid \cdot A_i$$

In Squeezer algorithm, the similarity measure given in Definition 6 is used to determine whether a tuple should be put into a cluster or not.

The Squeezer algorithm has n tuples as input and produce clusters as final results. Initially, the first tuple in the database is read in and a Cluster Structure (CS) is constructed with $C = \{1\}$. Then, the consequent tuples are read iteratively.

For every tuple, by the similarity function, we compute its similarities with all existing clusters, which are represented and embodied in the corresponding CSs. The largest value of similarity is selected out. If it is larger than the given threshold, denoted by $s$, the tuple will be put into the cluster that has the largest similarity value. The corresponding CS is also updated with the new tuple. If the above condition does not hold, a new cluster must be created with this tuple.

The algorithm continues until we have traversed all the tuples in the dataset.

The Squeezer algorithm is presented in Figure 1. It accepts as input the dataset $D$ and the user defined similarity threshold. The algorithm fetches tuples from $D$ iteratively.

Initially, the first tuple is read in, and the subfunction $addNewClusterStructure(\ )$ is used to establish a new Clustering Structure, which includes Summary and Cluster (Step 3-4).

For the consequent tuples, the similarity between each existing Cluster C and the tuple is computed using subfunction $simComputation(\ )$ (Step 6-7). Consequently, we get the maximal value of similarity (denoted by $sim_{\text{max}}$) and the corresponding index of Cluster (denoted by index) (Step 8-9). Then, if $sim_{\text{max}}$ is larger than the input threshold $s$, the subfunction $addTupleToCluster(\ )$ will be called to assign the tuple to the selected Cluster (Step 10-11). Otherwise, the subfunction $addNewClusterStructure(\ )$ will be called to construct a new CS (Step 12-13). Finally, outliers will be handled (Step 15), and the clustering results will be labeled on the disk (Step 16).

4. THE dSQUEEZER ALGORITHM AND usmSQUEEZER ALGORITHM

In this section, we will propose two algorithms, namely dSqueezes and usmSqueezes, by extending the Squeezer algorithm to domains with mixed numeric and categorical attributes.
4.1. The dSqueezer Algorithm

Since the Squeezer algorithm has been demonstrated to be very effective for clustering categorical datasets, in the dSqueezer algorithm, we adopt a simple strategy of transforming the original dataset into categorical dataset by discretizing numeric attributes. Then, the Squeezer algorithm is used to cluster the transformed dataset.

The dSqueezer algorithm is described in Figure 2. Techniques for the discretization of numeric attributes will not be discussed in this article because many algorithms are available. (See the work of Liu et al.\textsuperscript{18} for a more recent survey about the existing discretization methods.)

The computational complexity of the dSqueezer algorithm has two parts: (1) the complexity for discretizing the original dataset and (2) the complexity for

---

**Algorithm dSqueezer \((D, s)\)**

**Begin**

1. Discretizing the original dataset \(D\) into \(D'\)
2. Applying Squeezer \((D', s)\) on \(D'\)

**End**

---

**Algorithm Squeezer \((D, s)\)**

**Begin**

1. while \((D\) has unread tuple)\{
2. \hspace{1em} tuple = getCurrentTuple \((D)\)
3. \hspace{1em} if (tuple.tid == 1)
4. \hspace{2em} addNewClusterStructure \((tuple.tid)\)
5. \hspace{1em} else{
6. \hspace{2em} for each existed cluster \(C\)
7. \hspace{3em} simComputation \((C, tuple)\)
8. \hspace{2em} get the max value of similarity: \(\text{sim\_max}\)
9. \hspace{2em} get the corresponding Cluster Index: \(\text{index}\)
10. \hspace{2em} if \(\text{sim\_max} \geq s\)
11. \hspace{3em} addTupleToCluster \((tuple, \text{index})\)
12. \hspace{2em} else
13. \hspace{3em} addNewClusterStructure \((tuple.tid)\) \}
14. \}
15. handleOutliers()
16. outputClusteringResult()

**End**

---

Figure 1. The Squeezer algorithm.

Figure 2. The dSqueezer algorithm.
executing Squeezer algorithm on the transformed dataset. It is well established in Ref. 1 that the time complexity of part (2) is $O(n * k * m)$, where $n$ is the size of dataset, $m$ is the number of attributes, and $k$ is final number of clusters. As for part (1), discretizing the original dataset require at most two scans over the dataset, which is $O(n)$. Therefore, the overall complexity for the dSqueezer algorithm is $O(n * k * m)$.

The above analysis shows that the time complexity of dSqueezer algorithm is linear to the size of dataset, the number of attributes, and the final number of clusters, which make this algorithm more scalable.

### 4.2. The usmSqueezer Algorithm

For the usmSqueezer algorithm, we define a unified similarity measure for mixed-type attributes, in which both numeric and categorical attributes can be handled equally in the framework of Squeezer algorithm.

Let $A^n_1, A^n_2, \ldots, A^n_p, A^n_{p+1}, \ldots, A^n_m$ be a set of attributes with domains $D_1, \ldots, D_m$, respectively. Without loss of generality, we assume that the first $p$ elements are numeric attributes and the rest are categorical attributes.

**Definition 7.** For each numeric attribute $A^n_i$ ($1 \leq i \leq p$), the maximal attribute value of $A^n_i$ is denoted by $\max(A^n_i)$, and the minimal attribute value of $A^n_i$ is denoted by $\min(A^n_i)$. The interval of $A^n_i$ is defined as $\text{Interval}(A^n_i) = \max(A^n_i) - \min(A^n_i)$.

To handle both numeric and categorical attributes in the Squeezer framework, some modifications must be made to the Cluster Structure in our previous definitions.

**Definition 8 (Modified Summary).** Given a Cluster $C$, the mSummary for $C$ is defined as

$$m\text{Summary} = \{c_i | 1 \leq i \leq p\} \cup \{VS_i | p + 1 \leq i \leq m\}$$

where $c_i = (1/|C|) \sum_{j=1}^{|C|} b_{ij}$, $b_{ij}$ is the attribute value of $A^n_i$ and $VS_i = \{(a_j, \text{Sup}(a_j)) | a_j \in \text{VAL}_i(C)\}$.

In Definition 8, we extend the original Summary structure in the following way: For categorical attributes, the pairs of attribute value and support value are still maintained just as in Squeezer; for numeric attributes, their mean values are stored. With the Modified Summary, we can store the summarized cluster representation for data cluster with mixed types of attributes.

**Definition 9 (Modified Cluster Structure, MCS).** Given a Cluster $C$, the Modified Cluster Structure (MCS) for $C$ is defined as $CS = \{\text{Cluster, mSummary}\}$.
**Definition 10** (Unified Similarity Measure). Given a Cluster $C$ and a tuple $t$ with $tid \in TID$, the similarity between $C$ and $tid$ is defined as

\[
usmSim(C, tid) = \sum_{i=1}^{p} \frac{|a_i - c_i|}{Interval(A_i)} + \sum_{i=p+1}^{m} \left( \frac{\text{Sup}(a_i)}{|C|} \right)
\]

where $a_i = tid \cdot A_i$

**Lemma 1.** Given a Cluster $C$ and a tuple $t$ with $tid \in TID$, we have $0 \leq \frac{|a_i - c_i|}{Interval(A_i)} \leq 1$, where $a_i = tid \cdot A_i$ and $c_i$ is the mean value of attribute values of $A_i$ in $C$.

**Proof.** Trivial.

Lemma 1 shows that the contribution of each numeric attribute to the similarity value is normalized between 0 and 1. In other words, in the new proposed similarity measure, the numeric attribute and the categorical attribute could be treated equally. Hence, the proposed similarity measure avoids biased treatment of different kinds of attributes

**Lemma 2.** Given a Cluster $C$ and a tuple $t$ with $tid \in TID$, then $0 \leq usmSim(C, tid) \leq m$ holds.

**Proof.** Trivial.

From Lemma 2, it is easy to know that using the new similarity measure, we can cluster the dataset with mixed types of attributes in the similar way as we have done in Squeezer.

The $usmSqueezer$ algorithm is described in Figure 3. The $usmSqueezer$ algorithm scans the dataset twice. In the first scan, for each numeric attribute $A_i(1 \leq i \leq p)$, the maximal attribute value of $A_i: \max(A_i)$, and the minimal attribute value of $A_i: \min(A_i)$ are discovered (Step 1-5). Consequently, the interval of $A_i: Interval(A_i)$ is computed as $\max(A_i) - \min(A_i)$ (Step 6-8). In the second pass over the dataset, we use the unified similarity measure in the Squeezer algorithm framework to cluster mixed types of attributes (Step 9).

Similar to the analysis for the $dSqueezer$ algorithm, the time complexity of the $usmSqueezer$ algorithm is still $O(n \cdot k \cdot m)$, where $n$ is the size of dataset, $m$ is the number of attributes, and $k$ is final number of clusters.

**5. EXPERIMENTAL RESULTS**

Our goal for performing empirical studies with $dSqueezer$ and $usmSqueezer$ was twofold: (1) compare the clustering performance of our algorithms with existing clustering algorithms and (2) show the scalability of the two algorithms. To achieve this, we used two kinds of data: (1) the real datasets from the UCI Machine Learning Repository and (2) the artificially generated data, with mixed numeric and categorical attributes.
5.1. Real World Datasets

Our experiments with real datasets focus on comparing the quality of the clustering results produced by dSqueezer and usmSqueezer with other algorithms, such as k-prototypes. We choose datasets based on the consideration that they contain approximate equal numbers of categorical attributes and numeric attributes.

The first dataset was the credit approval dataset. The dataset has 690 instances, each being described by 6 numeric and 9 categorical attributes. The instances were classified into two classes, approved labeled as "/H11001" and rejected labeled as "/H11002." We compare our algorithm with k-prototypes algorithm; however, the k-prototypes algorithm cannot handle missing values in numeric attributes, and 24 instances with missing values in numeric attributes were removed. Therefore, only 666 instances were used.

The second dataset was cleve dataset, which has 303 instances, each being described by 6 numeric and 8 categorical attributes. The instances were also classified into two classes, each class is either healthy (buff) or with heart-disease (sick). The cleve dataset has 5 missing values in numeric attributes, all of them are replaced with the value "0."

Since the dSqueezer algorithm needs to discretize the numeric attributes in its preprocessing step, we disretized the numeric attributes in both of the above datasets using the automatic discretization functionality provided by the CBA software.20

---

**Algorithm** usmSqueezer(D,s)

begin
01 foreach record r in the dataset do begin
02 foreach numeric attribute Aᵢ do begin
03 update max(Aᵢ) and min(Aᵢ)
04 end
05 end
06 foreach numeric attribute Aᵢ do begin
07 Interval(Aᵢ) = max(Aᵢ) - min(Aᵢ)
08 end
09 Clustering D using Squeezer strategy with the unified similarity measure
end

---

Figure 3. The usmSqueezer algorithm.

---

1084 HE, XU, AND DENG
In addition, the clustering accuracy for measuring the clustering results is computed as follows, as has been done in Ref. 15. Suppose the final number of clusters is \( k \), clustering accuracy \( r \) is defined as:

\[
r = \frac{\sum_{i=1}^{k} a_i}{n},
\]

where \( n \) is number of instances in the dataset, \( a_i \) is number of instances occurring in both cluster \( i \) and its corresponding class, which has the maximal value. In other words, \( a_i \) is the number of instances with class label that dominate cluster \( i \). Consequently, the clustering error is defined as \( e = 1 - r \).

5.2. Clustering Results

We used the \textit{dSqueezer}, \textit{usmSqueezer}, and \textit{k}-prototypes algorithms to cluster the credit approval dataset and the \textit{cleve} dataset into different numbers of clusters, varying from 2 to 9. For each fixed number of clusters, the clustering errors of different algorithms were compared.

For the \textit{k}-prototypes algorithm, just as suggested in Ref. 15, all numeric attributes are rescaled to the range of \([0, 1]\). For the \textit{dSqueezer} and \textit{usmSqueezer} algorithms, the similarity threshold \( s \) is assigned to different values to get desired number of clusters.

Figure 4 shows the results on the credit approval dataset of different clustering algorithms. From Figure 4, we can summarize the relative performance of those algorithms as Table I.

That is, compared to the \textit{k}-prototypes algorithm, the \textit{dSqueezer} algorithm performs the best for three cases and the second best for four cases, and the \textit{usmSqueezer} algorithm performs the best for five cases and the second best for three cases. They only perform the worst in two and one cases separately. Furthermore, the average clustering errors of our algorithms are smaller than that of the \textit{k}-prototypes algorithm.

The experimental results on the \textit{cleve} dataset are described in Figure 5, and the summarization on the relative performance of the 3 algorithms is given in Table II.
From Figure 5 and Table II, the average clustering performance of our algorithms are better than that of the $k$-prototypes algorithm. At the same time, the cases that our algorithms performing the best and the second best also beat that of the $k$-prototypes algorithm.

The above experimental results demonstrate the effectiveness of $dSqueezer$ and $usmSqueezer$ for clustering dataset with mixed attributes. In addition, both of them outperform the $k$-prototypes algorithm with respect to clustering accuracy.

### 5.3. Scalability Test

The purpose of this experiment was to test the scalability of the $dSqueezer$ and $usmSqueezer$ algorithm when clustering very large datasets.

The German dataset in the UCI Machine Learning Repository has 1,000 instances, each being described by 7 numeric and 13 categorical attributes. The instances were also classified into two classes, each class is either good or bad. The synthetic datasets are concatenations of copies of the German dataset, the numbers of instances of the artificial datasets varying from 100,000 to 1,000,000.

Our algorithms were implemented in Java. All experiments were conducted on a Sun Ultra-Spare Workstation with 128 M of RAM and running Solaris 2.6. The $k$-prototypes program used here is implemented in C (which is provided by Dr. Joshua Huang), thus it is faster than its implementation in JAVA and the comparison is convincible.

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Average clustering error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-prototypes</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0.258</td>
</tr>
<tr>
<td>$dSqueezer$</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0.229</td>
</tr>
<tr>
<td>$usmSqueezer$</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0.217</td>
</tr>
</tbody>
</table>

Table I. Relative performance of different clustering algorithms (credit approval dataset).

![Figure 5. Clustering error versus different number of clusters (cleve dataset).](image-url)
For the dSqueezer and usmSqueezer algorithms, similarity thresholds are fixed to be the values that result in two clusters in the original German dataset, and the clustering results are saved to the hard disk. For the k-prototypes algorithm, the final number of clusters is set to two and all numeric attributes are rescaled to the range of $[0.1]$. The final clustering results are also saved to the disk.

Figure 6 shows the results of the three algorithms on clustering different numbers of objects into two clusters. When the number of records goes up to 900,000, the $k$-prototypes run out of memory.

One important observation from Figure 6 was that the running times of our two algorithms increase linearly as the number of records is increased. This feature is very critical in data mining applications. Furthermore, our algorithms run much faster than the $k$-prototypes algorithm, especially when the number of records is very large.

Table II. Relative performance of different clustering algorithms (cleve dataset).

<table>
<thead>
<tr>
<th>Ranking</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Average clustering error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-prototypes</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0.190</td>
</tr>
<tr>
<td>dSqueezer</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0.177</td>
</tr>
<tr>
<td>usmSqueezer</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>0.186</td>
</tr>
</tbody>
</table>

Figure 6. The execution time versus different number of tuples.
6. CONCLUSIONS

In this article, we consider the problem of clustering datasets with mixed types of attributes. Two efficient algorithms, namely dSqueezer and usmSqueezer, are proposed that extend the Squeezer algorithm to domains with mixed numeric and categorical attributes. Experimental results on real life and synthetic datasets show that our algorithms beat the popular \(k\)-prototypes algorithm with respect to both clustering accuracy and execution time.

For future work, we will take the proposed algorithms as the background-clustering algorithm for detecting cluster based local outliers\(^{21}\) and class outliers\(^{22}\) in large database environment with mixed-type attributes.

Acknowledgments

We would like to express our gratitude for Dr. Joshua Zhexue Huang of the University of Hong Kong for sending us the source codes of \(k\)-prototypes. We would also like to thank Danish Irfan for his help on proofreading. This work was supported by The High Technology Research and Development Program of China (Grant No. 2003AA422170, Grant No. 2003AA413021), the National Nature Science Foundation of China (Grant No. 40301038), and the IBM SUR Research Fund.

References


15. Huang Z. Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min Knowl Discov 1998;2:283–304.


