Clustering Sequences of Complex Objects

A. Ketterlin
LSIIT, URA CNRS 1871,
7, rue Descartes, F-67084 Strasbourg Cedex
e-mail: alain@dpt-info.u-strasbg.fr

Abstract

This paper is about the unsupervised discovery of patterns in sequences of composite objects. A composite object may be described as a sequence of other, simpler data. In such cases, not only the nature of the components is important, but also the order in which these components appear. The present work studies the problem of generalizing sequences of complex objects. A formal definition of generalized sequences is given, and an algorithm is derived. Because of the excessive computational complexity of this algorithm, a heuristic version is described. This algorithm is then integrated in a general-purpose clustering algorithm. The result is a knowledge discovery system which is able to analyze any structured database on the base of a unified, unsupervised mechanism.

Introduction

The process of knowledge discovery in databases has many facets. This paper explores one of them, namely the task of automatically discovering patterns in sequential data. The goal is to find classes of data which appear to evolve "in the same way". The most important aspects are that the algorithm is unsupervised (i.e., it doesn't require any teaching), and that the data to be analyzed may be structurally complex (i.e., may contain sub-level data). This paper describes an algorithm that learns classes of sequences of objects, and is organized as follows: the next section explains what sequences are, and how classes of sequences are described. The paper proceeds by briefly describing a general-purpose clustering algorithm, and how it is adapted to handle sequences of objects. Finally, some conclusions are drawn, and extensions of the approach are sketched.

Sequences of Complex Objects

The purpose of this section is to describe the kind of data the algorithm deals with, and then to explain how generalization is performed on such data.

Sequential Data

The problem of sequential data analysis has been studied in several contexts. Statisticians usually talk about *time series*, but most research in this area deals with the problem of *prediction*, i.e., predict the next event in the sequence given a certain number of its predecessors. Also, time series are usually bound to represent numerical or discrete simple quantities. A similar approach has been undertaken in the machine learning community, where it is usually called *discrete sequence prediction* (Laird & Saul 1994). In both cases, the goal is to predict a forthcoming event. A third, more recent approach, is the *frequent episodes discovery* approach (Mannila, Toivonen, & Verkamo 1995): the problem here is to find frequently appearing sequences given a stream of events labeled with a finite number of discrete symbols. This last approach is much more similar to the one described in this paper, which is to *cluster* whole sequences.

The field of knowledge discovery in databases explicitly aims at handling *databases*, in contrast with *data sets*. Databases are defined according to a *schema*, which details how pieces of information are organized and linked to each other. Data are usually structured when described at several levels of *abstraction*. Structured data types are usually described with the help of several *constructors*. The most common one is the *tuple* constructor, which allows one to describe a type of objects with several attributes. Another common constructor is the *set* constructor: an attribute's value may be a set of objects. Some clustering algorithms are able to build classes of such objects (Ketterlin, Gançarski, & Korczak 1995). This paper is dedicated to a third type of constructor: the *sequence* constructor, which allows an attribute to accept an ordered sequence of subobjects as its value.

The clustering algorithm described below works on objects which are tuples of values, some of these values being sequences of (sub-)objects. Let us briefly introduce an example in which sequential data appear. Imagine an agent (e.g., a robot) moving in some delimited space. Suppose now that the position of this agent is measured at sev-

eral times. The result is a sequence of positions, where a position might be a complex object, recording not only the physical position, but also other parameters, like for example, the temperature or any other relevant characteristic. The complex object here may be called a trajectory, and a potential data model could be:

```
position = tuple < x,y:  number;
                   t  :  number; ...>
trajectory = tuple <
   traj :  sequence < p :  position >; ...>
```

In this description, tuple and sequence stand for the basic "type constructors", and number stand for a basic data type; position and trajectory are complex data type names; and x, y, t, traj and p are attribute names. Two particular objects of such a type could be:

T_1 = < traj = [p_{11}, p_{12}, p_{13}] , ... >
T_2 = < traj = [p_{21}, p_{22}, p_{23}, p_{24}] , ... >

where, for example, the p_{11} instance of position may be described by:

p_{11} = < x = 25, y = 5 , t = 40 , ... >

The basic problem is: given several objects T_1, T_2... how is it possible to find classes of similar sequence?

Generalizing Sequential Data

Our main goal is to describe a system which is able to cluster complex objects, like the ones presented at the end of the previous section. These objects have "components" which are themselves objects, and may thus also be clustered. In fact, this remark is the basis of the algorithm presented in this paper. The fundamental argument is the following: it is not possible to cluster complex objects without first clustering their components. Two remarks can be made in favor of this argument:

- Sub-objects may themselves be complex, structured objects. One may imagine sequences of sets of any kind of objects. This means that it is not possible, in general, to consider the full description of components while clustering their containers;
- Classes of complex objects have to be represented as intensional descriptions, rather than as a set of members. Thus, to build classes of complex objects, one needs some "vocabulary" to formulate these classes. Names of classes of components are good candidates in this respect.

Another remark, which is of particular importance in a knowledge discovery context, is the following:

- The resulting classes have to be easily understandable by a human expert. This is especially true in unsupervised learning tasks, where very few, if any, objective measures are available to evaluate the results of an experiment. Abstraction levels (i.e., data types) which have been judged relevant at the modeling step have to be "preserved" at the analysis step, so as to help the interpretation of the results.

For all these reasons, the system described in this paper proceeds in a bottom-up way in terms of abstraction. In our previous example, this means that classes of positions will be built first, and these classes will be used to build classes of trajectories.

To understand how the clustering process works, let us first make some general hypothesis about the clustering algorithm. We will assume that a clustering algorithm is available for simple objects (i.e., objects described with simple—so-called "primitive"—attributes: numeric and nominal ones). We will also assume that this algorithm is able to produce a hierarchical clustering from simple objects. Finally, we will assume that there is some way to quantify the "precision" of each of the produced classes, i.e., that very narrow (or *specific*) classes have a higher score than wide (or *general*) ones. In our example, this implies that there exists an algorithm that will build a hierarchy of classes of positions. These classes will be used to describe classes of trajectories.

The next phase is to define a generalization mechanism for sequences of sub-objects. The question now is the following: given two sequences, what is the best description for what is common to both sequences, i.e., the best generalization. When computing this generalization, classes of components will be used in the following way: since each component is classified, it can be replaced by the name of the most specific class covering it. Hence, we transform a sequence of sub-objects into a sequence of class labels. Because these classes are hierarchically organized, the generalization process is much less strict than if a finite set of unordered class labels was used. Here is the basic problem we are left with: given two sequences U and V of class labels taken from a hierarchy $\mathcal{H}$, find a most specific generalization S covering both U and V.

The formal definition of the generalization is as follows: a generalization $S = [s_1, s_2, \ldots, s_n]$ of two sequences $U = [u_1, \ldots, u_\ell]$ and $V = [v_1, \ldots, v_m]$ must be such that:

1. $\exists\, f_u : [1,\ell] \to [1,n]$ (resp. $f_v : [1,m] \to [1,n]$) a surjective mapping such that $s_{f_u(i)}$ covers u_i (resp. $s_{f_v(j)}$ covers v_j);
2. $\forall i, j \in [1,\ell]$ (resp. $\in [1,m]$), $i < j$ implies $f_u(i) \leq f_u(j)$ (resp. $f_v(i) \leq f_v(j)$);
3. $s_k = \text{Gen}(\{u_i, i \in f_u^{-1}(k)\}, \{v_j, j \in f_v^{-1}(k)\})$ (recall that f_u and f_v are surjective mappings, so $f_u^{-1}(i)$ and $f_v^{-1}(i)$ are sets of indices), where $\text{Gen}(\gamma_1, \gamma_2, \ldots)$ denotes the most specific class covering $\gamma_1, \gamma_2, \ldots$
4. $\forall i \in [1, n-1], s_i \cap s_{i+1} = \emptyset$.

The first condition states that any element of U or V is "represented" in S. The second ensures that the order is preserved, and implies that several elements of a sequence may be mapped onto the same element of the generalization. The third expresses the fact that an element of S is the label of the most specific class covering the elements of U and V it represents. The fourth condition states that two adjacent class labels in the generalization are disjoint, i.e., cannot both match with the same element of one of the initial sequences. This condition enforces the uniqueness of the generalization. These conditions have an intuitive meaning: the generalization of two sequences is a sequence whose elements are generalizations of the elements of the initial sequences. These generalizations appear in the same order as in the initial sequences, with no overlapping between adjacent class labels.

Searching for Generalizations

The problem now is: "how does one compute such a generalization?" Unfortunately, there is no simple answer to this question. We are left with a typical matching problem. It is quite easy to show that an exhaustive search through the space of generalizations has an exponential cost in the length of the sequences. The only solution left is to perform a guided search through this space. Fortunately, there are some good heuristics to help that search. Before examining them, let us first give the overall matching algorithm. It simply consists in testing two limit cases:

- If one of the sequences U and V is of length 1, both sequences "collapse" into a generalized sequence of length 1. The class label used as the only element of the sequence is the label of the most specific class covering all elements of the initial sequences;
- If both sequences are of length 2, the resulting sequence is the superimposition of both initial sequences: the result of matching $[u_1,u_2]$ with $[v_1,v_2]$ is the sequence $[s_1,s_2]$ with $s_1 = \text{Gen}(u_1,v_1)$ and $s_2 = \text{Gen}(u_2,v_2)$. If s_1 and s_2 overlap, the result is $[\text{Gen}(s_1,s_2)]$. This is an example of explicit enforcement of the disjointness condition.

In any other case, the algorithm proceeds to test some pairing between one element of the first sequence and one element of the second sequence. The function PAIRS(U,V) returns all possible such pairings. For each couple of indices (i,j), it then solves the problem "on the left part" of the pairing, then on the "right part", and finally "pastes" together both partial results. The PASTE recombines both parts, taking into account that both have a common element (the class covering u_i and v_j). The final result of the matching is the best generalization found: the ranking is performed according to the Π heuristics, which will be described in the next section. The algorithm is:

```
GENERALIZE : U, V : Sequence ⟶ S : Sequence
    if |U| = 1 or |V| = 1 then
        return COLLAPSE(U,V)
    else if |U| = 2 and |V| = 2 then
        return SUPERIMPOSE(U,V)
    else
        for each (i,j) ∈ PAIRS(U,V) do
            let L = GENERALIZE(U[1,i],V[1,j])
            let R = GENERALIZE(U[i,ℓ],V[j,m])
            let R_{i,j} = PASTE(L,R)
        done
        return the R_{i,j} with best Π(R_{i,j})
    endif
```

The exhaustive algorithm is obtained by making the function PAIRS return all possible couples (i,j) (except $(1,1)$ and (l,m) to avoid infinite recursion). But as we have seen, this leads to an intractable algorithm. Fortunately, there are some ways to reduce this cost.

First, each couple (i,j) makes the element $\text{Gen}(u_i,v_j)$ appear in the resulting sequence. But since the quality of this class label can be quantified, couples can be ranked according to the quality of the class label they lead to. This allows for all standard heuristic searches: for instance, beam-search is achieved by sorting the list of couples and keeping only a fixed number of the best ones for further inspection. Second, because adjacent class labels in the resulting sequence must represent disjoint classes, the input sequences can be reduced after having been matched. In some circumstances, this leads to a dramatic decrease in their length, and allows for a rapid elimination of bad pairings.

The experiments conducted bu the author have shown that these simple heuristics allow keeping the execution times under a reasonable limit. More subtle optimizations can be applied, which lead to even better results (Ketterlin 1997).

The Learning Algorithm

Unsupervised Learning

The COBWEB algorithm (Fisher 1987) takes as input a list of objects and builds a hierarchy of classes grouping these objects. It proceeds by taking one object at a time, and drives it down the tree. At each level, the algorithm looks at several variations of the current level's partitioning. The selection of one of these variations is made on the base of a heuristics (decribed below), and the algorithm eventually recurses to the next level. Full descriptions of the algorithm are available elsewhere (Fisher 1987; Gennari, Langley, & Fisher 1989) for readers interested in the details. COBWEB has two major interesting aspects with respect to sequential data clustering. First, this algorithm is incremental: it never requires to find a generalization of more than two sequences, since the basic action is to put an object (a sequence) into a class (another sequence). Sec-

ond, the heuristics used by COBWEB is easily extendable: the "quality" of a partitioning is directly related to the quality of the individual classes in the partitioning, which in turn can be easily quantified.

One last positive aspect of COBWEB is its genericity: in fact, in the process of clustering sequential data, *two* class hierarchies have to be built. The first one groups the elements of the sequences, and the second one groups the sequences themselves. It is an important advantage to be able to use the same algorithm for *both* phases, because it reduces the overall complexity of the knowledge discovery system, and also because data can be analyzed at any level of structural complexity. One can imagine clustering sequences of sequences of whatever complex objects, for instance.

Handling Sequences

We still have to describe precisely how COBWEB makes its decisions. At the level of a class C, the algorithm has to decide how to modify the current set of sub-classes of C. To measure each potential partitioning $\{C_1, \ldots, C_K\}$, COBWEB uses:

$$\frac{1}{K} \sum_{k=1}^{K} P(C_k) \left(\Pi(C_k) - \Pi(C)\right)$$

where $P(C_k)$ measures the proportion of objects covered by C_k, and Π measures the individual precision of a class: this heuristics simply averages the gain in precision from the common super-class to each sub-class. We thus have to define Π for generalizations of sequences of sub-objects. Since classes of components (taken in the $\mathcal{H}$ hierarchy), can be evaluated with Π, evaluating a class of sequences consists in evaluating the classes of components whose labels are used in the generalization of the sequences. If the class S of sequences is described with the generalization $[\alpha_1, \alpha_2, \ldots, \alpha_n]$, the algorithm simply averages the individual classes' quality, namely:

$$\Pi(S) = \frac{1}{n} \sum_{i=1}^{n} \Pi(\alpha_i)$$

This is enough to select between different partitionings, and allows COBWEB to work on sequences of components.

Conclusion

This paper describes an algorithm which is able to cluster sequences of complex, structured objects—full details and examples are provided in (Ketterlin 1997). The process is based on a generalization procedure, which determines intensional representations of classes of sequences. This process is straightforwardly embedded into a general purpose clustering algorithm. This algorithm is able to cluster simple and complex objects: the extensions presented in this paper, along with previous work by the same author (Ketterlin, Gançarski, & Korczak 1995) on set-valued attributes, lead to a knowledge discovery system that can handle any type of composite objects.

There are several potential applications of clustering in the context of knowledge discovery in databases. The first, most obvious one, is to uncover important regularities in the database. Since similar data are put together, important trends may appear. A second, less immediate application is schema evolution. Since the algorithm forms a hierarchy of classes based on generalization, it may be applied to the contents of a database to suggest conceptual modifications, by eliciting classes present in the data, so as to help redesigning a conceptual model from data.

However, several extensions of the approach presented in this paper may be worth studying. The first one is to allow some tolerance during the generalization phase: as it is described in this paper, generalization requires a strict matching to be found between two sequences. Allowing some deviation would make the algorithm more robust. Another source of further developments is the fact that building generalizations is a process that allows other kinds of inference. One notable example of such inferences may be called *sequence completion*. Starting with an incomplete sequence (i.e., a sequence which is known to miss some elements), it is possible to use the class hierarchy to suggest classes of the "unknown" part(s) of the sequence. Such a completion procedure would make the clustering algorithm be also an information retrieval systems.

References

Fisher, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Machine learning* 2:139–172.

Gennari, J. H.; Langley, P.; and Fisher, D. H. 1989. Models of incremental concept formation. *Artificial intelligence* 40:11–61.

Ketterlin, A.; Gançarski, P.; and Korczak, J. J. 1995. Conceptual clustering in structured databases: a practical approach. In Fayyad, U. M., and Uthurusamy, R., eds., *Proceedings of the first international KDD conference*. Montreal, Canada: AAAI/MIT Press.

Ketterlin, A. 1997. Clustering complex objects: the case of sequences. Rapport de recherche, LSIIT, Université L. Pasteur, Strasbourg, France. (Available from `http://dpt-info.u-strasbg.fr/~alain`).

Laird, P., and Saul, R. 1994. Discrete sequence prediction and its application. *Machine learning* 15:43–68.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1995. Discovering frequent episodes in sequences. In Fayyad, U. M., and Uthurusamy, R., eds., *Proceedings of the first international KDD conference*. Montreal, Canada: AAAI/MIT Press.