Conceptual Clustering in Structured Databases: a Practical Approach

A. Ketterlin, P. Gançarski & J.J. Korczak
LSIIT, (CNRS, URA 1871) – Université Louis Pasteur
7, rue René Descartes, F-67084 Strasbourg Cedex.
e-mail: {alain,gancars,jjk}@dpt-info.u-strasbg.fr

Abstract

Many machine-learning (either supervised or unsupervised) techniques assume that data present themselves in an attribute-value form. But this formalism is largely insufficient to account for many applications. Therefore, much of the ongoing research now focuses on first-order learning systems. But complex formalisms lead to high computational complexities. On the other hand, most of the currently installed databases have been designed according to a formalism known as *entity-relationship*, and usually implemented on a relational database management system. This formalism is far less complex than first-order logic, but much more expressive than attribute-value lists. In that context, the database schema defines an abstraction space, and learning must occur at each level of abstraction. This paper describes a clustering system able to discover useful groupings in structured databases. It is based in the COBWEB algorithm, to which it adds the ability to cluster structured objects.

Introduction

Knowledge discovery in databases (KDD) is a current trend in machine learning research aimed at developing techniques and algorithms able to discover previously unknown knowledge in real-world databases (*i.e.* usually huge and noisy repositories of data) (Piatetsky-Shapiro & Frawley 1991). Clustering techniques respond in many ways to these requirements. The goal of clustering is to find important regularities in the data, and its result is a set of classes that have been found to accurately summarize the data.

In this paper, we concentrate on the fact that real-world databases are usually structured, *i.e.* that several levels of abstraction exist, at which the data are observed. Since many statistical as well as machine-learning algorithms assume that raw data appears in the form of a rectangular array, with individuals described along a fixed list of attributes, databases often need to be preprocessed to be analyzed. This paper takes an opposite position, by adapting algorithms to deal with the data they will meet.

Early attempts to handle more complex formalisms include (Michalski & Stepp 1983), where an extension of propositional logic is used. More recent works extend clustering techniques to higher level languages: KBG (Bisson 1992) deals with first-order logic. KLUSTER (Kietz & Morik 1994) uses a KL-ONE-*like* language to avoid computational complexity and still keep comfortable representative power. However, two main characteristics of real world databases may make existing algorithms hard to apply. First, many domains include a lot of numerical information, which often needs *ad-hoc* handling in logic-based formalisms. Second, as KDD concentrates on large databases, most current techniques (which work in a batch manner) may fail. In this paper, incrementality is considered a mandatory feature of a KDD system.

Also, most currently used database design formalisms are still far less complex than logic-based formalisms. In the field of database, at the user level, representation formalisms have more or less cristalized on the Entity-Relationship model, even though other promising approaches emerge. This consensus is partly due to the wide availability of relational databases management systems, on which E/R models can easily be implemented.

This paper advocates a compromise between representational complexity and efficiency. The next section explain how databases are usually structured and what form of data a clustering algorithm may be presented with. Then, a clustering algorithm is briefly reviewed, and an adaptation is proposed to handle structured data. An experiment illustrates the algorithm. Finally, a comparison with other system is sketched, and various aspects are discussed.

Database Modeling

Any real-world database is designed to reflect the physical organization of the domain in which it is used: therefore, databases are usually structured. The structure is derived by a preliminary analysis of the part of

the world to be represented. The result of this analysis is called a "conceptual model". This model defines several levels of abstractions at which data are observed.

The most widely used approach to this modeling task is called *Entity-Relationship modeling* (Chen 1976). Databases are usually designed at the E/R level, and then "projected" onto a more specific model (like the relational model) for the purpose of implementation (Teorey, Yang, & Fry 1986). In some commercial products, this process is even automated. Modern databases systems also include some "reverse-engineering" tools, which derive an E/R model from an existing relational database, thus allowing maintenance to operate at the E/R level.

The E/R modeling formalism is based on three notions:

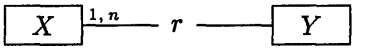
- *Entities*: an entity is basically the definition of the representation of a set of instances (occurrences) that will be represented with the same list of variables. Entities correspond to "types" of objects encountered in the domain, each one providing a particular level of abstraction.

- *Attributes*: attributes are the variables along which entities are described. Attributes may have two usages: identifiers (*i.e.* with distinct values along each occurrence) or descriptive attributes. Obviously, identifiers are of no use to a learning system, and only descriptive attributes may be used in search for useful generalizations.

- *Relationships*: a relationship connects two (or more) entities. Relationships are used to represent the structural organization of pieces of data, and provide links between distinct abstraction levels. A relationship is characterized by its "cardinality", which gives the number of occurrences of one entity connected to one occurrence of the other entity involved. *One-to-one* relationships connect one occurrence of one entity to exactly one occurrence of the other entity. *One-to-many* relationships allows several occurrences of the entity at the "many-side" to be connected to the same occurrence of the entity at the "one-side". *Many-to-many* relationships allows multiple connections in both directions.

Though one-to-one relationships may be seen as structuring conventions only, one-to-many and many-to-many relationships cannot be represented in attribute-value formalisms. The representation of an occurrence is no more of a fixed size, because the number of other occurrences to which it is connected to may not be the same for all occurrences: the data do no more fit in a rectangular array. If the E/R model is to be implemented on a relational database management system, two major cases can be distinguished: either some tables are augmented with *foreign-key* columns, or specific tables are created to contain occurrences of relationships (Teorey, Yang, & Fry 1986).

The "conceptual" model of a database is the result of a thorough study of the domain. From a learning point of view, it provides an abstraction space. Unsupervised learning in an E/R modeled domain involves learning at different levels of abstraction: this means that clusters have to be build for each entity. This problem has been called *structured concept formation* (Thompson & Langley 1991). To illustrate the problem, suppose two entities X and Y connected by a one-to-many relationship r (from X to Y).



Each occurrence x_i of X "contains" (more precisely "is connected to") any number of occurrences of Y. Objects like x_i are called *composite*, or simply structured, because their complete representation spans several levels of abstraction. They may be written down as:

$$x_i = \langle r = \{y_{i1}, y_{i2}, \ldots, y_{in}\}\rangle$$

Now the question is: If we had a set $x_1, x_2, \ldots x_n$ of occurrences of X, how would we find coherent groupings of them, and how would we compute a representation for a grouping? This is the topic of the next section.

Clustering and Composite Objects

Conceptual Clustering

Let us first briefly recall the algorithm used to solve the clustering problem. The COBWEB algorithm (Fisher 1987) takes as input a sequence of attribute-value lists, and forms a hierarchy of increasingly specific clusters. Unlike statistical cluster-analysis techniques, the description of a cluster is more than the set of instances it covers. In the case of COBWEB, some prototypical value is kept for each attribute, along with an indication on how instances' values may vary. Each incoming object is driven through the existing hierarchy in a top-down manner. On its path from the root to a leaf, the object may trigger restructuring operators, leading to topological modifications of the hierarchy.

The heuristic used to drive the incorporation of a new object is based on the predictive ability of the clusters. The predictivity of a cluster is an averaging of the predictivity of individual attributes inside that cluster. The predictivity of an attribute given a cluster is expressed as a summation of squared conditional probabilities of potential values of that attribute.

Let I be the number of attributes. The predictivity of a cluster C is then defined as:

$$\Pi(C) = \frac{1}{I} \sum_{i=1}^{I} \Pi(A_i, C),$$

where the predictivity of an attribute is

$$\Pi(A_i, C) = \sum_{j=1}^{J(i)} P(A_i = V_{ij} \mid C)^2,$$

if $J(i)$ is the number of potential values of A_i. This measure has been developed by cognitive scientists to account for human abilities in deciding which objects form a category. Space is insufficient here to repeat the argument, but full details are given by Fisher (1987). This predictivity measure is also the one used in CART, where it is called *gini-index*. The quantity $\Pi(C)$ may be interpreted as the number of attribute values that can be accurately predicted given membership to C. This gives a quantitative index that monotonically increases on any path from the root to a leaf of the hierarchy.

The predictivity of clusters is used to compute the quality of partitioning a cluster C into $\{C_1, \ldots, C_K\}$. Such a partition leads to a gain in predictivity, which can be evaluated with:

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

that is, a weighted average of the gain in predictivity between C and each of its sub-cluster. This heuristic is very similar to the one used in supervised learning algorithm like CART or ID3, except that the whole representation is used to measure the "quality" of the sub-clusters, instead of a single class-attribute.

The original COBWEB algorithm was designed to deal with nominal attributes. In that case, the predictivity of an attribute is computed from the frequency count of each value of each attribute, which is stored in each cluster. The CLASSIT algorithm (Gennari, Langley, & Fisher 1989) extends this formalism to deal with numerical attributes, where an underlying normal distribution is assumed. The predictivity can thus be expressed as the inverse of the standard deviation. Any new type of attribute could be defined, as soon as a representation is provided for generalizations, along with a measure of predictivity (*i.e.* $\Pi(A_i, C)$) to evaluate such generalizations.

Representation

As seen above, a preliminary analysis of the domain leads to entities, which represent abstractions of the data to analyze. In this section, we will concentrate on the problem of clustering occurrences of an entity which is at the "one-side" of a one-to-many relationship. We will assume that a hierarchical clustering of occurrences of the entity at the "many-side" can be computed, or is given as background knowledge. Occurrences of the entity at the "one-side" are called *composite* objects, and occurrences of the entity at the "many-side" are called *components*. The goal is to build a hierarchical organization of clusters of composite objects.

In contrast with strictly logical formalisms, which characterize clusters (or *concepts*) by necessary and sufficient conditions, COBWEB represents clusters in a probabilist form. The characterization of a cluster includes, for each attribute, a distribution of the values observed in instances covered by the cluster. A distribution includes a typical value and some information on how this value may vary between instances. Therefore, in the case where objects are structured, the requirements are twofold:

- find a characterization of a cluster of composite objects (*i.e.* a generalization for a set of sets of components);
- evaluate how much this characterization is "predictive" (*i.e.* how precisely components of instances of that cluster can be predicted).

The system described in this paper performs composite objects clustering in a "component first" style. Components are clustered first, leading to a component-clusters hierarchy. Then, composite objects are clustered. The characterizations of composite-clusters are built up with references to the component-clusters that describe commonalities between the covered composite objects.

Central Clusters

This section describes how, with the help of a component cluster hierarchy, the algorithm can build a composite cluster hierarchy. The COBWEB algorithm is used to cluster components as well as composite objects. This section describes the representation of composite clusters (*i.e.* the representation of a prototype) and their evaluation (*i.e.* the computation of $\Pi(C)$).

When presented with several sets of objects, the only way to give a general description (a *generalization*) of these sets is to find their common parts, *i.e.* the way they are similar to each others. The problem is thus to find their mutual intersection. But a crude intersection is of no use, because the instance space is usually large (if not infinite), and the intersection is usually empty. The idea behind structured concept formation is that

a preliminary clustering at the level of the components provides the "vocabulary" to express such generalizations in a flexible way.

The component clusters to be used in the description of a composite cluster are called *central clusters*, and must be such that:

1. each central cluster must cover at least one member of each value: this means that each cluster must characterize the intersection between all the sets (*i.e.* composite objects) under consideration;

2. all the members of the sets must be covered by central clusters: this means that central clusters must cover all the elements of all the sets, thus avoiding "marginal" intersections between only parts of the sets (that would leave some other components, *i.e.* members, uncovered);

3. central clusters have to be as specific as possible: this means that the set of central clusters is the most "precise" characterization of the intersection between all the sets.

Since the algorithm seeks a characterization of the intersection of all the sets of components, the first constraint is obviously derived. Note that at least one central cluster always exists: the root of the component cluster hierarchy covers all the components of all the composite objects, and thus can be used as a "worst-case" definition of a composite cluster. But obviously, the most informative characterization is the most specific one.

Once characterized by a set of component clusters, a composite cluster has to be evaluated. This evaluation must quantify the predictive ability of the set of central clusters, *i.e.* the precision with which one could infer the components of a composite object, given that it is an instance of the cluster. Since component clusters are discovered by COBWEB, they are hierarchically organized and labeled with their predictive ability (*i.e.* $\Pi(C)$), which monotonically increases along the paths from the root to the leaves. The predictivity of the set of central clusters is directly related to the individual predictivity of its members. But since all the central clusters do not cover the same number of components, their contribution to the overall predictivity is weighted according to their representativeness. Suppose A_c is the structured attribute, C is the composite cluster to evaluate, $\gamma_1, \ldots, \gamma_L$ are the central clusters. Our implementation uses:

$$\Pi(A_c, C) = \sum_{l=1}^{L} \frac{n_o(\gamma_l)}{N_o} \Pi(\gamma_l)$$

where $n_o(\gamma_l)$ is the number of components covered by γ_l, N_o being the total number of components in the objects covered by C. The weight associated with each central cluster may become important in the case of composite objects with large differences in their number of components. It has almost no effect when composite objects have approximately the same size (thus reducing the formula to an unweighted average of predictivity scores).

Properties

Incrementality The COBWEB algorithm is incremental, *i.e.* it accepts instances at any time and incorporates them in its memory structure. This is especially important when the system has to deal with very large, continuously growing databases, which is typically the case in a KDD system (even though this is seldom a primary focus of machine learning algorithms). Any extension to COBWEB should preserve incrementality to ensure its wide applicability.

The problem of incrementality may be stated as: given a composite cluster C, described with central clusters $\gamma_1, \ldots, \gamma_L$, and an incoming composite object O with components $O_1, \ldots, O_N$, is it possible to compute the set of central clusters of $C+O$ (*i.e.* the cluster obtained by incorporating O into C). Updates have to be made to the set of central clusters in two cases:

1. a central cluster does not cover at least one component of the incoming object;

2. a component of the new object is not covered by any of the current central clusters.

In the first case, one central cluster must be replaced by its most specific ancestor, and this generalization eventually repeated until reaching an ancestor that covers at least one component of the new object. In the second case, a new central cluster must be added: this cluster must cover the new object and at least one of the previous central clusters. Each time a central cluster is added and/or generalized, care must be taken that mutual exclusiveness is preserved (*i.e.* no two central cluster have a non-empty intersection). Greater details of the updating process are given in (Ketterlin, Gançarski, & Korczak 1995).

Complexity The COBWEB algorithm has a computational complexity of $O(N \log N)$, where N is the number of objects. The extension described in the previous section entails an additional cost due to the incorporation process. Let us consider a composite-cluster C covering N objects $O_1, \ldots, O_N$, and let $n_i = \mid O_i \mid$. It is easy to see that C will be represented by at most $\omega = \min_i\{n_i\}$ central clusters. Hence, the integration

of the next composite object in C may lead to at most ω generalizations during the first step of the updating process. The second step may apply only $\mid O_{N+1} \mid$ times. The whole updating process is thus linear in the average number of components per composite object. Incorporating N composite objects has a cost of $O(N^2 \log N)$ (to which must be added the cost of incorporating the components).

An Experiment

This experiment involves some simplified form of pattern recognition. The basic idea is to consider a pattern as a set of pixels. To compensate the loss of the information of the spatial arrangement of pixels, the representation of each individual pixel is based on several convolutions.

The design of the experiment is as follows: each pattern is computed from an underlying polygon, to which a rotation is applied, whose angle is randomly drawn. The rotated figure is then discretized, leading to a grey-level icon. Each icon is convolved with the Laplacian of a Gaussian at several different scales. Each non-white pixel in the original pattern is considered as a component of that pattern. Each component is described by the values of the convolutions at its position. These values can be seen as local characteristics of the pixel in the original pattern: in fact, as shown in (Marr & Hildreth 1980), convolution by the Laplacian of a Gaussian can be used as an edge detector (a value near zero meaning that the pixel is on an edge in the original image). In the experiments reported here, three convolutions are used, with σ respectively equal to 5/2, 3 and 7/2 (see (Marr & Hildreth 1980) for details about the meaning of this parameter). The convolutions are directly applied to the iconic images. Thus, in that case, the system has to build clusters of occurrences of the "Pattern" entity, expressed in terms of clusters of occurrences of the "Pixel" entity.

Three "patterns" were used to generate the data. Each pattern was randomly rotated by three different angles, and each time expressed as a set of pixels. Figure 1 shows the patterns, with the angle of rotation and the number of pixels (*i.e.* of components). An E/R model is shown in Figure 2.

The system thus has 426 components and 9 composite objects to cluster. The result of the clustering is shown on Figure 3. Results suggest that the set clustering algorithm is able to discover rotational-invariant classes of patterns.

Even though this experiment is far from being convincing from a pattern recognition point of view, it illustrates some of the properties of the algorithm described above. First, objects with different numbers of components are clustered. Second, a relatively large number of components are present in each object. Finally, this example illustrates the difference between distinct levels of abstraction: the hierarchy of clusters of pixels carries almost only numerical information, while the hierarchy of patterns exhibits some symbolic knowledge of simple patterns.

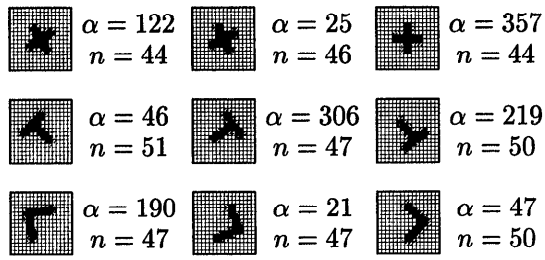


Figure 1: Nine rotated polygons.

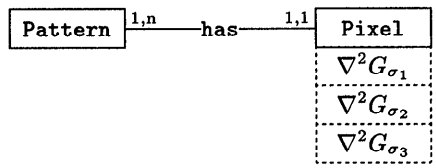


Figure 2: An E/R model of the pattern recognition domain: `Pattern` and `Pixel` are the entities, `has` is a one-to-many relationship (attributes are shown in dashed boxes).

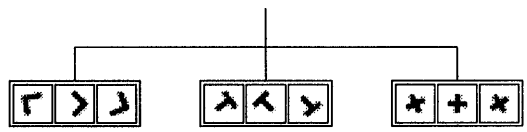


Figure 3: Results in the pattern recognition domain.

Discussion

Related Work

Most work on structural (also called relational) learning has been developed in a supervised setting (this includes "inductive logic programming"), and can hardly be used in unsupervised learning. However, some systems address the clustering problem.

LABYRINTH (Thompson & Langley 1991) is also based on COBWEB, and works approximately the same way as the algorithm described above. But it puts additional restrictions on representable instances, in the sense that all observations must have the same number of components. Since it does not use the composite clusters hierarchy as a "component-matcher", the process of adding an object to a cluster has a cost exponential in the number of components (all the possible

bindings between components and a predefined list of attributes are tested).

KBG (Bisson 1992) and KLUSTER (Kietz & Morik 1994) both employ high-level languages (respectively first-order logic and description logic). Both systems build a DAG of clusters, instead of a hierarchy. But both work bottom-up, and do not support incrementality. This may be prohibitive with large databases.

Limitations

Like any incremental system, COBWEB suffers from ordering effects, *i.e.* the discovered clusters may differ depending on the order in which the objects appeared. Clustering at higher levels of abstraction may be strongly affected by ordering effects at lower levels. This ordering problem is inherent to the algorithm, and is due to the hill-climbing search strategy. However, large data sets, like the one KDD is intended to, reduce the impact of order. Some authors have addressed this problem: a review, along with new solutions, appear in (Fisher 1995). These solutions are independent of the extensions we propose, and could therefore be directly integrated in a finalized system.

Because of the "component-first" strategy, the system described in this paper requires the E/R model to be put in the form of a directed acyclic graph (DAG). This DAG is then traversed bottom-up. But some models cannot be transformed into a DAG, because they exhibit cyclic relationships. In such cases, the algorithm cannot be applied without some sort of "bootstrapping". Note however that this problem is similar to discovering recursive predicates, and that similar problems are encountered in the field of knowledge representation, where so-called "terminological cycles" are, in some systems, simply not allowed.

Conclusion

An extension of the COBWEB clustering algorithm has been presented, which is applicable over one-to-many relationships. The algorithm builds a hierarchy of clusters, where each cluster is characterized by a prototype of its members. This prototype is expressed in terms of clusters of components that have themselves been discovered, *i.e.* generalizations are discovered at several levels of abstraction simultaneously. The algorithm applies whether all the composite objects have the same number of components or not. An example has been given to illustrate these abilities.

Planned further studies include the extension of the clustering mechanisms to ordered sequences of components. This goes a little beyond the E/R formalism, but is often implicitly incorporated (*e.g.* by the way of "date" attributes), and could model, for example, temporal variations of a component. Ordered lists are much more than sets, and additional constraints must be placed on the choice of central clusters. An implementation is currently under testing.

References

Bisson, G. 1992. Conceptual clustering in a first order logic representation. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, 458–462. J. Wiley and Sons.

Chen, P. P. 1976. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems* 1(1):9–36.

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

Fisher, D. H. 1995. Iterative optimization and simplification of hierarchical clusterings. Technical Report CS-95-01, Vanderbilt University, Nashville TN.

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. Hierarchical clustering of composite objects with a variable number of components. In *Working Papers of the Fifth International Workshop on Artificial Intelligence and Statistics.*

Kietz, J.-U., and Morik, K. 1994. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning* 14:193–217.

Marr, D., and Hildreth, E. 1980. Theory of edge detection. *Proceedings of the Royal Society of London* B. 207:187–217.

Michalski, R. S., and Stepp, R. E. 1983. Learning from observation: Conceptual clustering. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds., *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann.

Piatetsky-Shapiro, G., and Frawley, W. J. 1991. *Knowledge Discovery in Databases.* AAAI/MIT Press.

Teorey, T. J.; Yang, D.; and Fry, J. P. 1986. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys* 18(2):197–22.

Thompson, K., and Langley, P. 1991. Concept formation in structured domains. In Fisher, D. H.; Pazzani, M.; and Langley, P., eds., *Concept Formation: Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann.