A lattice-based approach to hierarchical clustering

Zdravko Markov
Department of Computer Science, Central Connecticut State University
1615 Stanley Street, New Britain, CT 06050
E-mail: markovz@ccsu.edu

Abstract

The paper presents an approach to hierarchical clustering based on the use of a least general generalization (lgg) operator to induce a lattice structure of clusters and a category utility objective function to evaluate the clustering quality. The objective function is integrated with a lattice-based distance measure into a bottom-up control strategy for clustering. Experiments with well-known datasets are discussed.

Introduction

In the context of Machine Learning clustering is an approach to discovering structure in data. Therefore the clusters are usually represented intensionally (in terms of relations, properties, features etc.) and hierarchical clustering techniques are of main interest. The term often used for this area is *conceptual clustering*. Three basic issues are important in conceptual clustering: the type of *cluster representation*, the *control strategy* used to search the space of possible clusterings and the *objective function* used to evaluate the quality of clustering. Clusters can be represented by necessary and sufficient conditions for cluster membership (e.g. rules, decision trees) or probabilistically – by specifying the probability distribution of attribute values for the members of each cluster. The control strategy determines the way in which the clustering tree is generated (e.g. top-down, bottom-up, hierarchical sorting). The objective function can be integrated in the control strategy or used separately. The classical conceptual clustering system COBWEB (Gennari, Langley, & Fisher, 1989) uses the former approach – the search in the space of possible clusterings is based on an objective function that evaluates the candidate clusterings. The latter approach is taken in another classical system, CLUSTER/2 (Michalski & Stepp, 1983), that generates an initial clustering and then optimizes it iteratively by using an objective function. A more recent approach along these lines is proposed in (Fisher, 1996).

Recently a distance-based approach to creating concept hierarchies has been proposed in (Markov, 2000). It has been described as a general framework for inductive learning. In this paper we further develop this approach for the purposes of conceptual clustering by integrating an objective function into its control strategy.

Basic algorithm

Given a set of examples (observations) the task of a hierarchical clustering system is to create a tree structure of clusters where the children of a cluster partition the examples covered by their parent. The algorithm proposed in (Markov, 2000) accomplishes this task by using a lattice-based distance measure and employing a bottom-up control strategy. Using a given set of examples E the algorithm builds a partial lattice structure G, where E is the set of all maximal elements of G. The algorithm works in agglomerative fashion and at each step selects pairs of observations/clusters to be merged in single clusters. The selection is based on a distance measure and the new cluster is produced by applying a least general generalization operator (lgg) on the selected pair. The lgg is based on a subsumption relation that is used next to remove the observations/clusters subsumed by the newly generated cluster. The algorithm works iteratively and at each step updates two sets: C – the set of top level clusters in the current hierarchy where the two clusters to be merged are selected from and G – the current hierarchy accumulating all instances and clusters generated by lgg. Formally the algorithm is as follows:

1. Initialization: $G = E$, $C = E$;
2. If $|C| = 1$ then exit;
3. $T = \{h | h = lgg(a,b), (a,b) = argmin_{a,b \in C}\ d(a,b)\}$;
4. $DC = \{a | a \in C \text{ and } \exists h \in T, h \preceq a\}$;
5. $C = C \setminus DC$;
6. $G = G \cup T$, $C = C \cup T$;
7. go to step 2.

Hereafter we limit our discussion to an *attribute-value* representation with *nominal attributes* for both examples (observations) and clusters. The partial ordering $\preceq$ in this representation is the set inclusion $\subseteq$. The *lgg*

(infimum) of two elements is defined as their intersection, i.e. $lgg(a,b) = a \cap b$. The distance function is defined as $d(a,b) = 2^{-cov(a)} + 2^{-cov(b)} - 2 \times 2^{-cov(a \cap b)}$, where $cov(a)$ is the number of examples covered by a, i.e. $cov(a) = |\{b | b \in E, a \preceq b\}|$.

Since in most cases the set T created in step 3 is not a singleton, a restricted version of the algorithm is usually used. The basic idea is instead of the whole set T to use just a single element of T (arbitrarily chosen). This reduces the computational complexity of the algorithm[1] and also narrows the created hierarchy.

We illustrate the algorithm by a simple example of hierarchical clustering of a set of 10 animals represented by 6 attributes: covering, milk, homeothermic, habitat, eggs and gills. Table 1 shows the initial set E and the resulting set G. To display the examples/clusters we use positional encoding, i.e. each element of the hierarchy is a list of values or underscores (denoting a missing attribute). In this representation the members of a cluster h are all elements of G that h subsumes, i.e. the elements that have the same values at the same positions as h, where an underscore in h matches all values. Obviously because of the random choice in step 3, the clusterings may differ from run to run. Figure 1 shows two possible hierarchies generated by the algorithm (the bottom hierarchy corresponds to the set G, shown in Table 1). We are further interested in the following questions: how can we evaluate these hierarchies and can we use this evaluation to improve the quality of clustering. We discuss these questions in the next section.

E	G
{feathers, f, t, land, t, f}	{feathers, f, t, land, t, f}
{hair, t, t, air, f, f}	{hair, t, t, air, f, f}
{feathers, f, t, air, t, f}	{feathers, f, t, air, t, f}
{hair, t, t, land, f, f}	{hair, t, t, land, f, f}
{scales, f, f, land, t, f}	{scales, f, f, land, t, f}
{none, f, f, land, t, f}	{none, f, f, land, t, f}
{none, t, t, sea, f, f}	{none, t, t, sea, f, f}
{hair, t, t, sea, t, f}	{hair, t, t, sea, t, f}
{scales, f, f, sea, t, f}	{scales, f, f, sea, t, f}
{scales, f, f, sea, t, t}	{scales, f, f, sea, t, t}
	{scales, f, f, sea, t, _}
	{_, f, f, land, t, f}
	{_, f, f, _, t, _}
	{hair, t, t, _, f, f}
	{feathers, f, t, _, t, f}
	{_, f, _, _, t, _}
	{hair, t, t, _, _, f}
	{_, t, t, _, _, f}
	{_, _, _, _, _, _}

Table 1: Attribute-value representation of observations and clusters for the "animals" example (t and f stand for true and false respectively)

Evaluating clustering quality

A commonly used measure to evaluate clustering quality is the *category utility* function proposed in (Gluck & Corter, 1985). This is the objective function used in COBWEB (Gennari et al., 1989) and in many related systems. Category utility attempts to maximize both the probability that two objects in the same category have attribute values in common and the probability that objects from different categories have different attribute values. The category utility function assigns a value $CU(C_k)$ to each cluster C_k as follows:

$$CU(C_k) = \sum_i \sum_j [P(A_i = V_{ij}|C_k)^2 - P(A_i = V_{ij})^2]$$

The sums are calculated for all attributes A_i and all their values V_{ij}. The probabilities $P(A_i = V_{ij}|C_k)$ and $P(A_i = V_{ij})$ are calculated as relative frequencies of the occurrence of the V_{ij} value for the A_i attribute within the cluster C_k and within the whole set of observations respectively.

$CU(C_k)$ estimates the quality of individual clusters. To measure the quality of a clustering we need a function that evaluates the quality of a partition of data.

[1]The complexity of the algorithm in this case is $O(n^3)$, where $n = |E|$. More details on this issue can be found in (Markov, 2000).

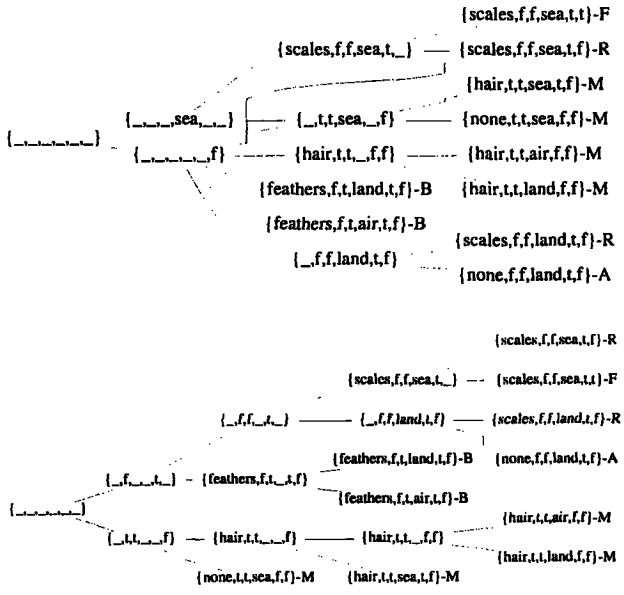

Figure 1: Concept hierarchies for the "animals" example

For this purpose the average category utility of clusters in the partition is used, i.e. $PU(\{C_1, C_2, ..., C_n\}) = \frac{1}{n}\sum_k CU(C_k)$.

In hierarchical clustering the *partition utility function* PU can be used at any level of the concept hierarchy. Applied to the children of the root PU evaluates the overall partition of the observations. For example the PU scores for the top level partitions of the two concept hierarchies in Figure 1 are as follows:

$PU(\{\{\_,\_,\_,sea,\_,\_\},\{\_,\_,\_,\_,\_,f\}\}) = 0.191$
$PU(\{\{\_,f,\_,\_,t,\_\},\{\_,t,t,\_,\_,f\}\}) = 0.612$

These PU scores show that the second clustering is better than the first one. This also becomes clear if we look at the class memberships of the observations. The classes that the observations belong to are shown at the leaves of the hierarchies as M, B, R, F and A (denoting mammal, bird, reptile, fish and amphibian respectively). These labels are *not* used by the algorithm, however they may be used to evaluate "externally" the created hierarchy. For the bottom clustering, $\{\_,t,t,\_,\_,f\}$ represents the cluster of mammals, and the other cluster – $\{\_,f,\_,\_,t,\_\}$, groups all non-mammals. The structure of $\{\_,f,\_,\_,t,\_\}$ reflects the natural way of grouping birds, reptiles, fish and amphibians. This observation is also supported by the PU scores of the lower level partitions. For example, the PU scores of the partitions of the mammal clusters in the two hierarchies are:

$PU(\{\_,\_,\_,\_,\_,f\}) = 0.374$
$PU(\{\_,t,t,\_,\_,f\}) = 0.421$

Another reason for these scores is that the mammal and non-mammal clusters in the first hierarchy overlap (they share 3 observations) while in the second hierarchy they are completely disjoined.

The above considerations suggest that there is a kind of propagation of "good" PU scores (i.e. good partitions of data) from the lower to the upper levels of the concept hierarchy. This in turn suggests a way to integrate the PU scores as an objective function into the control strategy of our basic algorithm described in Section 2. This algorithm uses a bottom-up control strategy. At each iteration a new cluster h is added to the hierarchy. Since at this stage the whole partition at that level of the hierarchy is still unknown, h should be evaluated *individually* with respect to its contribution to the quality of clustering for the partition in which it will be included at a later step. That is, h has to maximize its CU score. To implement this we modify step 4 of the basic algorithm accordingly:

4. $h = argmax_{h \in T}\ CU(h)$, $T = \{h\}$, $DC = \{a | a \in C, h \preceq a\}$;

An important issue in bottom-up clustering is how to determine *when to stop merging clusters.* In our algorithm the newly generated cluster h replaces the clusters/observations that it subsumes. Since the goal is to maximize the quality of clustering in terms of PU scores, it is expected that $CU(h) > PU(\{h_1, h_2, ..., h_n\})$, where $h_1, h_2, ..., h_n$ are the clusters/observations that h subsumes, i.e. $h \preceq h_i$, $i = 1, 2, ..., n$. This condition can be used to stop the process of merging clusters and thus to cut off the hierarchy at a certain level. For a greater cutoff, a stronger condition can be defined as $CU(h) > \max\{CU(h_1), CU(h_2), ..., CU(h_n)\})$. In fact, we introduce a cutoff parameter $W \in [0, 1]$, that is used to control this process by gradually switching between the two conditions: $CU(h) > PU(\{h_1, ..., h_n\}) + W * (\max\{CU(h_1), ..., CU(h_n)\} - PU(\{h_1, ..., h_n\})$.

The cutoff parameter W is used in step 4 of the algorithm in the following way. If h satisfies the stopping condition, then its successor with maximal CU score is used as a top-level cluster (an immediate successor of the root). This is achieved by excluding the latter and all its successors from the candidates for further merging.

Experiments

To evaluate the performance of the algorithm we conducted experiments with some well-known datasets. They include the previously discussed "animals" data containing 10 examples with 6 attributes, the "zoo" dataset – 59 examples with 16 attributes and the small soybean dataset – 47 examples with 35 attributes, all obtained from the UCI ML repository (Blake & Merz, 1998). The class attributes were removed from all three datasets and after the concept hierarchies were generated the class information was used as an additional, "external" criterion to evaluate the quality of clustering. The idea was that the top-level partition should reflect the class distribution in the set of observations. As a basis for comparison a version of the COBWEB algorithm provided by the WEKA-3 ML system (Witten & Frank, 1999) was used in the experiments. Table 2 shows the PU scores of the top level partition produced by two clustering algorithms – the lattice-based clustering with gain function (denoted as LGG-PU) and COBWEB with the three data sets discussed above. Table 3 shows the performance of the same algorithms on the same datasets measured by the entropy of the class attribute in the clusters from the top level partition. The entropy of the class attribute in a cluster C_k is calculated as follows:

$$H(C_k) = -\sum_i P(C = V_i)\log_2 P(C = V_i),$$

where V_i are the values of the class attribute C, that occur in the examples belonging to cluster C_k. Then the entropy in a partition $\{C_1, C_2, ..., C_k\}$, or the *partition entropy* PH is:

$$PH(\{C_1, C_2, ..., C_n\}) = \sum_k \frac{|C_k|}{|E|} H(C_k),$$

where the terms $\frac{|C_k|}{|E|}$ are weight coefficients that account for the cluster size.

	LGG-PU	COBWEB
animals (10 obs, 6 attrs)	0.505	0.505
zoo (59 obs, 16 attrs)	0.745	0.721
soybean (47 obs, 35 attrs)	0.75	1.46

Table 2: *PU* scores for the top level partition of three datasets

	LGG-PU	COBWEB
animals (10 obs, 6 attrs)	0.27/4	0.27/4
zoo (59 obs, 16 attrs)	1.03/7	0.99/4
soybean (47 obs, 35 attrs)	0/8	0.49/4

Table 3: *PH* scores/number of clusters for the top level partition of three datasets

In the ideal case each cluster contains examples of a single class and then the *PH* function of this partition is 0. In the general case small *PH* scores mean that the partition of the data corresponds to the distribution of the class attribute. Of course this also depends on the representativeness of the set of examples (i.e. whether it contains a sufficient number of instances from every class).

A further experiment was conducted with the relational version of the basic algorithm. As described in (Markov, 2000) this algorithm can handle relational data by using a relational lgg. For our representation such lgg can be implemented by *anti-unification*. This is an operation that replaces same constants with same variables. For example $lgg(\{a,b,a\},\{c,d,c\}) = \{X,Y,X\}$, where X and Y are variables. The main advantage of this representation is the possibility to define explicitly *equality* of attribute values. To investigate the behavior of our clustering algorithms we used the training sample of the MONK1 dataset (Thrun et al., 1991). This dataset describes a concept with 6 attributes by 61 positive and 61 negative examples. The propositional description of the target concept is:

```
{octagon, octagon, _, _, _, _}
{square, square, _, _, _, _}
{round, round, _, _, _, _}
{_, _, _, _, red, _}
```

While the relational one is:

```
{X, X, _, _, _, _}
{_, _, _, _, red, _}
```

For clustering we used just the positive examples. The whole set of examples (including the negatives) was used for "external" evaluation of the top level clusters. The experiments with the MONK1 data are summarized in Table 4.

The main findings reflected in the experiments are the following:

- For propositional data the LGG-PU algorithm performs similarly to COBWEB as long as the *PU* scores are concerned. In some cases LGG-PU achieves much better *PH* scores than COBWEB. A cutoff parameter W close to 0 results in high *PU* scores and low *PH* scores. With W close to 0, the *PU* scores are low and the *PH* scores are high.
- In the relational case both algorithms achieve 0 entropy at the top level partition. LGG-PU however built two clusters that match exactly the original definition of target concept. COBWEB created very fragmented definition obviously because it does not take into account the relational similarity between observations.
- In the relational example (MONK1) the *PU* scores are low for both algorithms. This is because the two top level concepts in the original definition of the target concept substantially overlap.

	PU	*PH*	# of clusters
COBWEB	0.14	0	7
LGG-PU	0.16	0	2

Table 4: Results of clustering the MONK1 data. The *PU* and *PH* scores, and the number of clusters refer to the top level partition. *PH* is calculated by using the whole set of examples (positive and negative).

Related work

Most of the approaches to hierarchical and conceptual clustering are related to two classical systems – CLUSTER/2 (Michalski & Stepp, 1983) and COBWEB (Gennari et al., 1989). CLUSTER/2 generates an initial clustering and then optimizes it iteratively by attempting to minimize the cluster overlapping. Clusters are represented as necessary and sufficient conditions for cluster membership (e.g. rules) and are derived by a standard concept learning algorithm. In COBWEB the clusters are represented probabilistically by the probability distribution of the attribute values for the members of each cluster. The control strategy used is based on the *PU* function. There are many other systems that follow the COBWEB's approach. Some of them elaborate the objective function by using information-based heuristics (e.g. the information gain used in decision tree induction), Bayesian variants of *PU* (used in AUTOCLASS (Cheeseman, Kelly, Self, Stutz, Taylor, & Freeman, 1988)), or the Minimal Message Length (MML) principle (used in SNOB (Wallace & Dowe, 1994)). Others combine the COBWEB and the CLUSTER/2 approaches - they first generate initial clustering and then optimize it by using various objective functions. The approach taken in (Fisher, 1996) is based on this idea. It uses hierarchical sorting to induce a clustering and then iteratively simplifies it by applying various techniques as redistribution, reordering etc.

There is third approach to hierarchical clustering that is based on lattice structures. The lattices are a useful

mathematical formalism that resembles cluster hierarchies. The problem with the use of lattices for clustering is that the techniques for generating clusters should have some formal properties (usually not present in the heuristic algorithms). The computational complexity of the lattice algorithms is also high. Nevertheless lattice approaches to clustering exist and some of them are successful. For example (Guenoche & Mechelen, 1993) use Galois lattices to induce hierarchical clusterings for binary attributes. This approach is based on the so-called maximal rectangles and employs some standard algorithms from Galois lattice theory.

Our approach is also based on lattices. Its theoretical basis is the distance measure introduced in (Markov, 2000) that allows to evaluate the similarity between examples/clusters as well as to apply consistently a generalization operation (lgg) to build the lattice structure. The use of a *PU*-based objective function allows to improve the quality of clustering and to reduce the complexity of the algorithm. The main features of the approach can be summarized as follows:

- LGG-PU represents clusters as propositional or relational rules. Similarly to CLUSTER/2 it allows cluster overlapping.
- It uses a *PU*-based objective function to evaluate the quality of clustering.
- The control strategy of the algorithm is a bottom-up greedy search based on a consistent integration of a lattice-based distance measure and a *PU*-based objective function.
- LGG-PU is able to build relational descriptions of the clusters. Although the *PU* function does not work well on relational data, it is still useful in the control strategy of the algorithm.

Conclusion

We described an approach to hierarchical clustering based on the use of a least general generalization (lgg) operator to induce a lattice structure of clusters and a category utility objective function to evaluate the clustering quality. The objective function is integrated with a lattice-based distance measure into a bottom-up control strategy for clustering. The preliminary experiments showed that the approach compares well with other similar approaches and in some cases outperforms them. The future work will address the following issues:

- Extending the algorithm to handle numeric attributes. This can be achieved by extending the partial ordering relation (presently set inclusion) and by defining a proper *lgg* that can generalize numeric attributes too. Then we can use the same distance measure and a *PU* function that uses probability distributions.
- Although the algorithm evaluates clusters locally, it also maximizes the overall clustering quality since the local evaluation goes through all levels of the hierarchy in a bottom-up fashion. Of course, as this is a greedy search, there is no guarantee that the global maximum can be found. With this respect other approaches to evaluating the quality of clustering have to be investigated too (e.g. the recursive ones used in COBWEB).

References

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases..

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AutoClass: a Bayesian classification system. In *Proceedings of the Fifth International Workshop on Machine Learning, Ann Arbor*, pp. 54–64 San Mateo, CA. Morgan Kaufmann.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research, 4*, 147–179.

Gennari, J. H., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence, 40*, 11–61.

Gluck, A., & Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of thr Cognitive Science Society*, pp. 283–287 Hillsdale. Lawrence Erlbaum.

Guenoche, A., & Mechelen, I. V. (1993). Galois approach to the induction of concepts. In Mechelen, I. V., Hampton, J., Michalski, R., & Theuns, P. (Eds.), *Categories and Concepts: Theoretical Views and Inductive Data Analysis*, pp. 287–308. Academic Press.

Markov, Z. (2000). An algebraic approach to inductive learning. In *Proceedings of 13th International FLAIRS Conference*, pp. 197–201 Orlando, Florida, May 22-25. AAAI Press, Extended version in IJAIT, Vol 10, No.1-2, to appear in March 2001.

Michalski, R., & Stepp, R. (1983). Learning from observation: conceptual clustering. In Michalski, R., Carbonell, J., & Mitchell, T. (Eds.), *Machine Learning: Artificial Intelligence Approach*, Vol. 1, pp. 331–363. Tioga.

Thrun, S. B., et al. (1991). The MONK's problems - a performance comparison of different learning algorithms. Tech. rep. CS-CMU-91-197, Carnegie Mellon University.

Wallace, C., & Dowe, D. (1994). Intrinsic classification by mml – the snob program. In *Proceedings of the Seventh Australian Joint Conference on Artificial Intelligence*, pp. 37–44 Armidale, Australia. World Scientific.

Witten, I., & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.