Concept Formation by Incremental Conceptual Clustering

Mirsad Hadzikadic
The University of North Carolina
Department of Computer Science
Charlotte, NC 28223

David Y. Y. Yun
Southern Methodist University
Dept. of Computer Science and Engineering
Dallas, TX 75275

Abstract

Incremental conceptual clustering is an important area of machine learning. It is concerned with summarizing data in a form of concept hierarchies, which will eventually ease the problem of knowledge acquisition for knowledge-based systems. In this paper we have described INC, a program that generates a hierarchy of concept descriptions incrementally. INC searches a space of classification hierarchies in both top-down and bottom-up fashion. The system was evaluated along four dimensions and tested in two domains: universities and countries.

1. Introduction

A system described in this paper, INC (INcremental Conceptual clustering system), *learns from observation* by recognizing regularities among events (objects, instances, examples, etc.) and organizing them into a hierarchy of concepts. Lebowitz [1987] described learning from observation as a task that is important in domains where events arc not pre-classified, but where one still wishes to detect general rules and intelligently organize events. Michalski and Stepp [1983] defined *conceptual clustering* as an important form of learning from observation, in which a configuration of events forms a class only if it is describable by a concept from a predefined concept class. A performance element then uses such concept descriptions to make inferences about new events based on partial information.

INC classifies events in the order of their appearance, thus building its knowledge base incrementally. It handles events in large numbers. In fact, the larger the number of events the better the hierarchy of concepts. In addition, INC's generalizations are pragmatic - they do not perfectly describe all the instances they cover. The concept representation mechanism implemented in INC is based on research done by Rosch and Mervis [1975], who have hypothesized that the members of categories (classes) which are considered most *prototypical* are those with the most attributes in common with other members of the category and the least attributes in common with other categories.

This work was mainly influenced by work of Michalski and Stepp [1983, 1986] and Fisher and Langley [1985] in conceptual clustering, similar incremental conceptual clustering systems COBWEB [Fisher, 1987], UNIMEM [Lebowitz, 1987], and CYRUS [Kolodner, 1984], and the system developed by Hadzikadic and Yun [1987,1988].

2. System Description

Input to INC is a series of events, given to the system one at a time. The system's task is to recognize patterns of similarity among events (if they exist) and represent their generalizations in a hierarchy of non-disjoint concept descriptions (although each event is stored under one subclass only).

2.1. Knowledge Representation

The evidence that *prototypes* play a critical role in human categorization is compelling. Rosch and Mervis [1975] have demonstrated the existence of prototypes for both natural categories (like colors and animals) and artificial categories (like dot patterns and schematic drawings). INC uses a *schema* formalism to implement prototypical representation of events and concepts. A schema is a declarative structure that organizes pieces of knowledge, related to the same entity, into a unitary whole. A template of the schema structure is given in table 1.

The *rel* (relevance) parameter represents the frequency of attribute-value pairs found in members' descriptions. At this point, the system supports both nominal and structured attributes. The *strength* parameter reflects how closely an event/class resembles its superordinate class, i.e., to what degree the event/class matches a prototypical representation of the concept.

2.1.1. Computing the Relevance Parameter

All attribute-value pairs from an event description participate in the concept description, once the class membership is determined. The more attribute-value pairs in the class description, the more general the description, for an event has more attribute-value combinations to

Table 1

name	schema-1
attributes	att-1, val-1, rel-1 . . . att-n, val-n, rel-n
special-of	sup-concept, strength
general-of	sub-concept-1, strength-1 . . . sub-concept-m, strength-m

"choose" from (match) to score above the predefined threshold and claim membership. However, the relevance of attribute-value pairs in class descriptions is generally lower than the corresponding relevance in event descriptions. Thus, fewer attribute-value pairs will score above the threshold and be included in calculating the strength parameter. The relevance *(rel)* of an attribute-value pair *(att.val)* in a description of a class *(c)* is computed as the sum of relevances of the corresponding attribute-value pairs in member descriptions (e_i,), divided by the total number of members (n), i.e. (equation 1):

$$rel(c, att, val) = \frac{\sum_{i=1}^{m} rel(e_i, att, val)}{n}$$

where $m < n$ stands for the number of events e_i *e* c with the property *(att.val).*

To improve the efficiency of the process, attribute-value pairs with a relevance below the threshold are temporarily dropped from a concept description (as long as their relevances remain low), since they do not contribute significantly to the prototypicality effect. Attribute-value pairs with a relevance of 1.0 are true of all members of the class, thus effectively implementing the inheritance property.

2.1.2. Computing the Strength Parameter

Once a class description has been generated and corresponding relevances calculated, the system can proceed to receive another event In order to decide whether to create a new class or place the event under one of the existing generalizations, INC computes the degree of class membership (similarity) with all top-level classes. The system first sums up the minimum of relevances for the attributes having the same value in descriptions of both the class (c) and the event *(e).* The resulting sum is then divided by the number of attribute-value pairs in the class description *(n).* This process is summarized in the following equation (2):

$$link(c, e) = \frac{\sum_{i=1}^{n} \min\{rel(c, att_i, val), rel(e, att_i, val)\}}{n}$$

where $link(c, e)$ stands for the strength of the link between the class *c* and the event *e,* and *rel(x,* att_i *val)* represents the relevance of the *(*att_i*val)* pair in *x.*

To determine which (if any) of the existing top-level prototypical class descriptions *D fa)* the new event *e* resembles the most, the system will compute *link* c_i, *e)* for $i = 1, \cdots, n,$ where *n* stands for the number of classes under consideration. The event then belongs to the class which both maximizes the value of *link* and is greater than the prespecified threshold (0.5 in our case). The corresponding class description is subsequently updated according to equation 1. This process is then repeated for the children of a chosen class etc., until the process reaches the leaves of the tree.

2.2. Operators

Similarly to COBWEB [Fisher, 1987], there are four operators used by INC to generate a hierarchy of concepts:

- *Place* - place the event into an existing class. First determine the best host and then calculate the new relevances of the attribute-value pairs in the class description according to equation 1.
- *Create* - create a new class. A description of the new class is exactly the same as a description of the event
- *Merge* - merge two or more classes into one. This operator is applied when there is more than one 'best' host for the event A new class is introduced which replaces all of the best hosts. The participating classes (best hosts) are stored as subclasses of the newly created class.
- *Split* - split the class into two or more classes. Where there is no good host for a given event, INC does not create a new class automatically. It first takes the best out of no-good hosts and evaluates its subclasses. If it finds a good host(s) among them, it replaces the best of no-good hosts with its subclasses, and the process continues.

2.3. Algorithm

A control strategy implemented in INC is summarized in the following procedure.

```
Procedure INC(Event, Root)
1. Update a description of Root with a description of Event
2. Compute the similarity between Event and Root's children.
3. IF a single best host is found,
   THEN update its description and call INC(Event, BestHost).
   ELSE
     IF more than one best host is found, THEN
       (a) merge the best hosts and Event,
       (b) update the description of the root of the newly generated subtree, and
       (c) compute the strength of the hierarchical links.
```

```
ELSE
  IF a good host is not found, THEN
    (a) evaluate (compute the similarity with) children
    of the best of no-good nodes, and
    (b) find the best host(s).
    IF found, THEN
      (a) split the best of no-good nodes into 2 classes:
      (1) one that contains Event and the best of no-
      good nodes' children similar to Event, and (2) a
      class that contains all other children of the best
      of no-good nodes;
      (b) update the class descriptions, and
      (c) compute the strength of hierarchical links.
    ELSE
      (a) create a new singleton class for Event, and
      (b) compute the strength of the hierarchical link.
```

Given an event and a (root of) hierarchy that summarizes previously seen events, INC first updates a description of the root with a description of the event, and then computes the similarity (according to equation 2) between the event and all the children (subclasses) of the root. If there is only one class (best host) similar to the event (scoring above the *s* threshold - set to 0.5), then INC updates the best host's description and invokes the same procedure recursively, with the root being replaced by the best host. In general, the system will try to "push" an event down the tree as far as possible in order to place it in the most specific subclass.

If there is more than one best host for the event, INC replaces them (along with the event) with a new generalized class, generates its description, creates the links (pointers) from the best hosts as well as the event to the newly generated class and computes their strengths, thus effectively merging them into the same class. When discovering best hosts, the system requires that only the most similar class "scores" above the *s* threshold. All other best hosts need to be within *e % (e* is another threshold in the system) of the most similar class, with respect to its degree of similarity to the event. We have been experimenting with several values for the *e* threshold in 0-40 range. The optimal value depends on the values assigned to other thresholds in the system. The value we used most often for the *e* threshold was 10 (actually implemented as 0.1).

However, if a good host is not found, then the system computes the similarity between the event and the children of the best of "no-good" nodes and tries again to establish the best host(s). If the effort is successful, then the best of no-good nodes is split into two classes. First class will contain the event and the best of no-good nodes' children similar (enough) to the event Second class will be comprised of the rest of the children of the best of no-good nodes. The class descriptions are then updated and the strength of the hierarchical links computed.

Finally, if a best host was still not found, then INC creates a new singleton class (for the event) and computes the strength of the link to the superordinate class.

In general, INC carries out a hill-climbing search through a space of hierarchical classification schemes. The system uses operators that let it search in both directions (place and create vs. merge and split). The merge and split operators provide the system with a form of backtracking to help it "correct" some of the earlier mistakes. The events are stored under generalizations that describe them best Although an event can be stored under only one generalization, it may match descriptions of two or more concepts, thus introducing a possibility for multiple membership. The resulting concept hierarchy is used by the performance system to make inferences about new, previously unseen events.

3. Experiments

INC was implemented in Common Lisp and evaluated on SUN workstations. It has been tested in two domains: countries and universities. The domain of universities will be described in this paper in greater detail. The following attributes were used in descriptions of 45 universities[1]: state, location, control, no-of-students-thous, male/female, student/faculty, sat-verbal, sat-math, expenses-thous, percent-financial-aid, no-applicant-thous, percent-admittance, percent-enrolled, academics-env, social-env, quality-of-life, and academic-emphasis. Not all attributes had to be defined in an event description. The relevance of an attribute-value pair in the event description was presumed to be 1.0, which was consistent with our assumption that an event represents a singleton class.

4. Evaluating the System

We have used four dimensions to evaluate a performance of the system. The first dimension covers the effect of varying the values of the three thresholds defined in the system: $, *e* and *d* thresholds. Different distributions of threshold values yields (somewhat) different concept hierarchies.

The second dimension is defined as the time needed to insert a new event into an existing hierarchy. It is evaluated with respect to the number of instances presented to the system prior to the current event.

The third dimension used to evaluate the system's performance is defined as the time needed to retrieve a "correct" concept for a given event, as well as the percentage of correct retrievals with respect to the number of total attempts.

Finally, the forth dimension is defined as the percentage of correct classifications of "unseen" events given an existing hierarchy of concepts, thus effectively implement-

[1] Provided by M. Lebowilz.

ing a performance component of the system.

4.1. Varying the Threshold Values

Initially, we adopted the following values for the three thresholds defined in the system: s=0.5, e=0.1, d=0.2. It meant that:

(a) In order to be similar to a concept, an event had to match at least half of the attribute-value pairs used in the concept description (or any combination of them resulting in a cumulative score of at least 0.5, 1.0 being a maximum) => s threshold.

(b) An event was declared similar to more than one concept if the degrees of similarity between those concepts and the event did not differ from the highest degree of similarity (max), achieved between the event and one of the concepts under consideration, for more than 10% of max => e threshold.

(c) Attribute-value pairs with a relevance below 0.2 in a concept description were not taken into consideration when computing the degree of similarity between an event and the concept => d threshold. However, these attribute-value pairs were considered during the process of *updating* the concept description.

In order to evaluate the effect of varying threshold values on the resulting classification, we fixed the value for s threshold and vary the s and d thresholds in 0-0.4 range. Two interesting conclusions were drawn from this experiment*

(1) Distributions {s=0.5, e=0.25, d =0.1} and {s=0.5, e=0.4, d=0.07 have produced identical hierarchies, although they have differed in the amount of time needed to complete the classification process. It seems that changes in the value of e threshold can compensate for some changes in d threshold values. At the same time, those hierarchies were the simplest ones of all - they contained fewest number of classes.

(2) For some values of e threshold (e.g., 0.1), varying d threshold values did not make any difference. An explanation may be that if the value of e threshold is conservative, then infrequent attribute-value pairs do not contribute significantly to a resulting hierarchy. The situation, however, changed after we relaxed the value of e threshold to 0.25 or 0.4.

4.2. Efficiency Considerations

A representative of the hierarchies containing the fewest number of classes (9) was then evaluated in terms of its computational efficiency. The results are summarized in column 1 of table 2. Column E defines a position of an event in a sequence of events. The headings of other columns in the table include information about the values for the *s, e,* and d thresholds, respectively. The values provided in the table are given in milliseconds, and represent the time needed to insert i-th event into the hierarchy, given $i-1$ already classified events and a particular distribution of threshold values. This insertion time also includes the time needed to restructure a hierarchy when appropriate, as a result of the merge and split operators.

Table 2

E	1 0.5, 0.4, 0.1 (complete desc.)	2 0.5, 0.4, 0.1 (reduced desc.)	3 0.5, 0.1, 0.2
1	17	17	17
10	1800	1050	550
20	2966	1234	867
30	4034	1434	1150
40	5500	1966	1500

The values given in column 1 are obtained as a result of considering all attribute-value pairs of all concepts on the path to the most specific subclass similar to an event Pairs with a relevance lower than d threshold are evaluated but not taken into consideration when computing the degree of similarity. To determine the rate of speed-up achieved by completely removing attribute-value pairs with a relevance below d threshold from a concept description, we have modified the system accordingly and tested it for the same threshold values. The results of the testing are provided in column 2 of table 2. It is obvious that the gain in efficiency is significant However, the modified system will not perform well in situations where the external world is characterized by constant changes and transitions. A more adaptive system may be warranted in such circumstances.

Column 3 represents an example of a hierarchy with 12 classes. That example proved to be the most efficient classification tree. In general, a larger number of classes means both fewer merge operations and shorter paths, which improves the efficiency of the classification hierarchy. Consequently, there is a trade-off between two contradictory requirements: a smaller number of classes vs. reduced processing time.

4.3. Concept retrieval

The third and fourth dimensions are introduced to evaluate both the quality and the efficiency of generated concept descriptions. This section is concerned with evaluating the quality of concept descriptions through the number of correct retrievals, and evaluating the efficiency of a concept hierarchy through the time needed to retrieve given events/concepts.

The two best hierarchies from the previous section were taken as a basis for this phase of the evaluation process, and consequently compared according to the obtained results. We used all the events that participated in generating the two hierarchies to retrieve the most similar concept/event stored in the hierarchy. Ideally, we would like to see an event retrieve "itself. However, since the basis of our approach is a prototypical representation of concepts, other members of the class may change the concept description significantly, thus causing the event under consideration to appear not so "prototypical" of the concept under which it is stored. That will prevent the system to retrieve the correct event/concept in certain situations. Also, it is possible that the system retrieves the correct concept, but decides not to consider its instances if the degree of similarity between the event and the concept is below the threshold (0.5). The results of this phase of the evaluation process arc summarized in table 3.

Table 3

Distribution	Correct instance	Correct concept	Incorrect
0.5, 0.4, 0.1	30 (65%)	10 (22%)	6 (13%)
0.5, 0.1, 0.2	46 (100%)	0 (0%)	0 (0%)

It is obvious that the {s=0.5, e=0.1, d=0.2} hierarchy consists of higher quality concept descriptions that summarize given events better than the other classification being evaluated. The 100% retrieval accuracy (vs. 65% for the other case, or 87% – both correct instance and correct concept retrievals included) more than compensates for the 16% increase in retrieval time (1038 vs. 1201 milliseconds).

At this point, it is safe to conclude that our original evaluation criterion, which favors hierarchies with fewer classes, contradicts the criterion that favors concepts of higher quality. This introduces another trade-off which needs to be taken into consideration when implementing the system in a particular domain.

4.4. Classifying Unseen Events

Classifying unseen events is an inherently subjective procedure since the proper classification depends on the perception, goals, and relevant knowledge of the observer. To reduce this subjectiveness as much as possible, we have decided to introduce the following scenario: (a) take the best hierarchy generated so far as a referent hierarchy; (b) take at least half of the events that have participated in generating the referent classification and run the system again with those events as the input; (c) use the rest of the events (nonclassified ones) to retrieve the most similar instances/concepts; and (d) compare retrieved instances/concepts with the events' intended concepts, as defined by the referent hierarchy.

The scenario outlined above was implemented in the following way: the {s =0.5, e=0.1, *d=02}* hierarchy was accepted as a referent classification, 26 events were used to generate a new hierarchy, and the other 20 events were used as unseen instances. We have again distinguished between two cases: retrieving a concept and retrieving an instance of a concept. Ideally, the system will retrieve a concept, given an event that is an instance of the concept. However, sometimes a concept has not been created, and the system will retrieve another event (a singleton class) most similar to the input event These cases arc *potentially* correct classifications since the retrieved event either belongs to the same concept as the input event (in the referent hierarchy) or it would have belonged to the same concept had there not been other events to alter the prototypical description of the concept before the retrieved event was presented to the system.

As a result, the system correctly classified 95% of the input events (50% correct and 45% potentially correct classifications), while misclassifying only 5% of them.

5. Related Work

Three incremental concept formation systems - COBWEB [Fisher, 1987], UNIMEM [Lebowitz, 1987], and CYRUS [Kolodner, 1984] - have influenced our approach to conceptual clustering. COBWEB [Fisher, 1987] constructs a concept hierarchy from the top down to summarize instances described as sets of features. The system modifies its concept descriptions and hierarchy as it classifies each instance. COBWEB employs probabilistic representations and an explicit evaluation function (category utility) to determine optimal clusterings. UNIMEM [Lebowitz, 1987] is also an incremental conceptual clustering system. Its search through a space of hierarchies can be described as hill climbing. The system does not build its hierarchies in an entirely top-down or bottom-up fashion - it has operators for merging and deleting nodes and associated subtrees. The CYRUS system [Kolodner, 1984] makes use of domain knowledge to determine which elements of instances can best serve as discriminants among concepts, thus avoiding combinatorial explosions in retrieval and concept formation.

INC differs from the three systems mentioned above in several aspects: knowledge representation formalism, type of attributes supported, definition of operators used by the search procedure, clustering evaluation mechanism, and similarity function. The naturalness, flexibility, and simplicity of both the knowledge representation formalism and similarity function provide INC with a powerful mechanism for dealing with incomplete and inconsistent event descriptions: attributes and their values are not predefined, multiple values are allowed, some attributes may be missing, some incorrect values may be specified, etc. The relevance mechanism, similarity function, and control knowledge will eventually cause noisy attribute-value pairs to disappear from a concept description, as well as the "correct" pattern to emerge in the form of a cluster and its description. At the same time, the simplicity of computing the similarity function greatly improves the efficiency of the system. The efficiency can be increased even more by raising the value of *d* threshold. On the other hand, the greater the value of *d* threshold, the lower the quality of the concept description. Consequently, there is a limit to the degree of improvement that can be achieved by manipulating the value of the threshold.

However, INC pays the price for this simplicity of the similarity function: descriptions of a class and its subclasses significantly overlap, thus wasting a potentially significant amount of memory. This is especially true in the case of inherited properties (the ones with the relevance of 1.0).

6. Research Issues

There are several important directions in which this work can be extended. Currently, we are working on incorporating a GDN-like structure (Goal Dependency Network [Stepp and Michalski, 1986]) into the system. Also, we are adding information about the context of the problem-solving task as well. The information about a goal and a context of classification will help the system determine relevant attributes for discriminating among candidate concepts, both in retrieval and clustering processes. The final result will be of higher quality and more useful to the performance element. In addition, the goal and context information may help the system determine the value of *e* threshold. When crisp and well-specified classes are sought, the threshold value should be low and, vice versa, a high value of *e* threshold will encourage largely overlapping classes.

A well-defined similarity function is a key to a successful classification. The similarity function defined in this paper is a simple one, and certainly can be improved. Future research will pay considerable attention to that problem.

People often generate non-disjoint concepts. Although somewhat imprecise, those concepts offer flexibility in dealing with a complex external world. INC creates non-disjoint concept descriptions (an event description can match several concept descriptions), but it stores an event under only one concept. However, INC can easily be updated to store multiple copies of an event when appropriate by modifying the merge procedure. We are planning to explore and carefully evaluate this possibility as well.

7. Conclusion

Incremental, concept formation is an important area of machine learning. It is concerned with summarizing real-world information in a form of concept hierarchies, which will eventually ease the problem of knowledge acquisition for knowledge-based systems. In this paper we have described INC, a program that generates a hierarchy of concept descriptions incrementally. The system searches a space of classification hierarchies in both top-down and bottom-up fashion. We have evaluated INC along four dimensions: the effect of varying threshold values, the cost of inserting a new event into an existing hierarchy, the accuracy and cost of concept retrieval, and the success rate in classifying unseen events. The system was tested in two domains: countries and universities. We feel that INC represents a promising step toward systems that will be capable of constructing, maintaining, and refining a knowledge base automatically.

Acknowledgments

We thank M. Lebowitz for kindly supplying the experimental data used to evaluate the INC system.

References

[Fisher and Langley, 1985] Fisher, D. and Langley, P. Approaches to Conceptual Clustering. *Proceedings of the 9th IJCAI,* 691-697, Los Angeles, CA, 1985.

[Fisher, 1987] Fisher, D. Knowledge Acquisition Via Incremental Conceptual Clustering. In *Machine Learning,* 2, 2, 139-172,1987.

[Hadzikadic, 1987] Hadzikadic, M. Concept Formation by Heuristic Classification. *Doctoral Dissertation.* Southern Methodist University, Dallas, 1987.

[Hadzikadic and Yun, 1988] Hadzikadic, M. and Yun, D. Y. Y. Concept Formation by Goal-Driven, Context-Dependent Classification. *Proceedings of the 3rd International Symposium on Methodologies for Intelligent Systems,* 322-332, Torino, Italy, 1988.

[Kolodner, 1984] Kolodner, J. L. Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model. Lawrence Erlbaum Associates, Publishers, London, 1984.

[Lebowitz, 1987] Lebowitz, M. Experiments with Incremental Concept Formation: UNIMEM. In *Machine Learning,* 2,2,103-138,1987.

[Michalski and Stepp, 1983] Michalski, R. S., and Stepp, R. E. III. Learning From Observation: Conceptual Clustering. In *Machine Learning: An Artificial Intelligence Approach,* R. S. Michalski, J. G. Carbonell, and T M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.

[Rosch and Mervis, 1975] Rosch, E. and Mervis, C. B. Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology* 7 573-605,1975.

[Stepp and Michalski, 1986] Stepp, R. E. III, and Michalski, R. S. Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects. In *Machine Learning II: An Artificial Intelligence Approach,* R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.