Intelligent Information Systems 2006
New Trends in Intelligent Information Processing and Web Mining
Ustron, Poland, June 19-22, 2006

The Use of Compound Attributes in AQ Learning

Janusz Wojtusiak[1] and Ryszard S. Michalski[1,2]

[1] Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA, USA
[2] Institute of Computer Science
Polish Academy of Sciences
Warsaw, Poland

Abstract. Compound attributes are named groups of attributes that have been introduced in Attributional Calculus (AC) to facilitate learning descriptions of objects whose components are characterized by different subsets of attributes. The need for such descriptions appears in many practical applications. A method for handling compound attributes in AQ learning and testing is described and illustrated by examples.

1 Introduction

Attributional Calculus (AC) is a logic system that combines elements of propositional calculus, predicate calculus, and multi-valued logic for the purpose of facilitating *natural induction*, a form of machine learning whose goal is to induce hypotheses from data in the forms close to natural language descriptions [8]. One of the novel concepts introduced in AC is a *compound* attribute, which is useful for describing objects whose components are characterized by different subsets of attributes. Such objects occur in many practical problems, for example, in medicine and agriculture, where different body organs or plant parts are characterized by different attributes.

To describe such objects, predicate calculus can be used, as done in Inductive Logic Programming, but this would introduce a substantial complexity to the learning process. Compound attributes allow one to avoid this complexity while still enabling simple descriptions of objects consisting of different components. The use of compound attributes simplifies learning descriptions of complex objects, and increases comprehensibility of its results, because such descriptions correspond more closely to equivalent natural language expressions.

For a simple illustration of a compound attribute, consider a standard logic-style description of *weather: windy=yes* & *cloudy=yes* & *humid=yes.* Using a compound attribute, such a description would be equivalently expressed as: *weather: windy* & *cloudy* & *humid*, which is closer to the equivalent natural language description. In this example, "weather" is a compound attribute, and windy, cloudy, and humid are values of its constituent attributes.

This paper uses the following naming convention. If c is a compound attribute and x is its constituent attribute, then we express this as ***c.x***. If a compound attribute, c1, includes a constituent attribute, ***c2.y***, that is also compound, we write this as ***c1.c2.y***. It is also assumed that learned descriptions (hypotheses) are represented in form of attributional rulesets generated by AQ21's learning module [8,13].

2 Learning and Testing Compound Descriptions

The algorithm for learning concept descriptions with compound attributes ("compound descriptions" for short) uses operators for learning descriptions with standard attributes (described in Sections 1 and 2.2), and consists of the following steps:

L1 Convert compound attributes in the training examples into standard (non-compound) attributes
L2 Learn descriptions using standard attributes
L3 Convert learned descriptions into forms with compound attributes.

The testing and application of descriptions with compound attributes is done in two steps:

T1 Convert compound attributes into standard attributes both in testing data and in rules.
T2 Apply/test converted rules to the converted data.

Testing results are presented in form of summaries used in AQ learning, such as best match accuracy, correct match and precision [13].

2.1 A Brief Description of AQ Learning

The developed method for learning with compound attributes has been implemented in the newest version of the AQ21 learning program. Before describing this method, to make the paper self-contained, we start with a brief description of the AQ learning method implemented the AQ21 program.

Programs from the AQ family learn hypotheses in the form of sets of attributional rules. One form of an attributional rule is:

$$CONSEQUENT <= PREMISE |\_ \; EXCEPTION \quad (1)$$

where consequent, premise and exception are complexes, which are conjunctions of attributional conditions (a.k.a. selectors), where $|\_$ is an exception operator. Here is an example of such a rule: $[part = fitting] <= [weight = 2..5] \& [color = red\ v\ yellow] \& [height < 2] |\_ \; [width > 1]$ which means that a part is acceptable if its weight is between 2 and 5 (units are defined in the attribute domain), its color is red or yellow, and its height is less than 2, except for parts whose width is greater than 1.

Concept descriptions learned by AQ are in the form of attributional rulesets, defined as sets of attributional rules with the same consequent [10,8,13]. The main operator used in basic form of AQ is star generation, which given a positive example (a seed) and a set of negative examples, generates the set of all maximal generalizations of the positive example that do not cover negative examples (called the star of the seed). In order to prevent an exponential growth of the star size, AQ employs a beam search that at each step of star generation maintains only a group of generalizations (attributional rules) that have the highest score on a user-defined rule quality measure. AQ determines the best rule from the generated star, and stores it. A new seed is selected from the positive examples uncovered by the stored rules, and the process repeats until all positive examples are covered. Such an algorithm creates a ruleset that is complete and consistent with regard to the training data, providing that examples of different classes are distinct in the representation space. To cope with the noise in the data, the rule quality measure is relaxed to allow for rule inconsistency.

The rule quality measure is in the form of Lexicographical Evaluation Function (LEF) that combines different elementary criteria for rule preference [8,13]. By default, AQ21 uses LEF = $\{< MaxNewPositives, 0\% >$, $< MinNumSelectors, 0\% >$, $< MinCost, 0\% >\}$, which means that it seeks first rules that cover maximal number of new examples, then selects among them rules with minimum number of selectors, and finally selects the rule that has the lowest cost (a "0%" following each precondition indicates the tolerance with which the criterion is applied). The user can build different LEFs by choosing different elementary criteria from the predefined list [13].

Fig. 1 presents the basic AQ algorithm in a pseudocode. The input to the algorithm consists of a set of positive concept examples, P, a set of negative concept examples, N, and a multi-criterion measure of rule quality LEF.

```
HYPOTHESIS = NULL
While not all positive examples are covered by HYPOTHESIS
Select an uncovered positive example e+ and use it as a seed
Generate star G(e+, N)
Select the best rule R from the star according to LEF, and add it to HYPOTHESIS
```

Fig. 1. Pseudocode of the simplest version AQ algorithm.

More advanced versions of AQ include methods for coping with noise, learning approximate descriptions (patterns), learning rules with exceptions, constructive induction that improves the original representation space, and other features.

The AQ21 program used in this research works in two basic modes, one is "Theory Formation" that creates complete and consistent rulesets, and the second is "Pattern Discovery" that seeks the strongest patters that represent the best tradeoff between rule consistency and completeness e.g. [4]. It also includes multi-seed star generation, learning alternative hypotheses, reasoning with meta-values in data, automatic attribute selection, and learning rules with exceptions [13,9,11].

2.2 Application and Testing Modules

AQ21 implements several variants of two main rule testing programs, ATEST for event classification and EPIC for temporal episode classification. The AQ21's testing module allows a multiple event classification, meaning that a single event can be classified to several classes, if it matches them with a degree above a specified threshold. The motivation here is that is better to provide an imprecise answer than a wrong answer.

In the ATEST module each event can be matches strictly, with degree of match 0 or 1, or flexibly, where degree of match is a number between 0 and 1. A ruleset-event matching procedure works on three levels: matching of a single selector with an event, matching of a rule with the event, and matching of entire ruleset (disjunction of rules) with the event. Several aggregation methods for all three steps are described in [8,13]. The EPIC module includes an additional aggregation step, namely, matching of an episode (a temporal sequence of events) with sequence models.

As mentioned before, matching compound selectors is done by transforming them into conditions involving constituent attributes, and then using efficient, previously developed matching methods.

2.3 Conversion of Learned Rules into Compound Rules

Conversion of learned descriptions into compound descriptions is done in a way that maximizes their similarity to the equivalent natural language expressions, and by that improves their comprehensibility.

Each complex (conjunction of attributional conditions) is converted into a compound complex by grouping attributes. The grouping is done by putting together, in one selector, all values of the constituent attributes of the compound attribute. For example, a complex: $[head.color = red]\&[head.shape = square]$ is converted into a compound complex $[head : red\&square]$, which is shorter, and more directly corresponds to an equivalent natural language expression. Similarly, a complex: $[head.color = red\ v\ green]\&[head.shape = square]$ is converted into: $[head : (red\ v\ blue)\&square]$. To avoid parentheses in the last expression, it is rewritten in AQ21 into: $[head : red\ v\ blue\ color\&\ square\ shape]$, which facilitates its translation into a non-ambiguous natural language expression.

The above use of attribute names in addition to their values in compound selectors is done in AQ21 in the following situations:

- on the user request
- when values of a constituent attribute are linked by internal disjunction
- when domains of constituent attributes include the same values
- when a constituent attribute is continuous

The use of attribute names in selectors with continuous attributes is particularly important for their interpretability. For example, it is easier to interpret a selector in the form: $[head : 17\ size]$ ("the head is of size 17" than in the form $[head : 17]$.

2.4 Determining Coverage of Compound Selectors

By default, AQ21 displays different selector coverages when printing learned hypotheses. These coverages include: *support* (p), *negative support* (n), *unique support* (u), *confidence* (p/(p+n)), *cumulative support* (pc), *cumulative negative support* (nc), and *cumulative confidence* (pc/(pc+nc)), and *cumulative support* is defined as the number of positive examples covered by a condition (selector) and all selectors preceding it in the rule. *Cumulative negative* support is defined analogically.

Because compound selectors can be viewed as products of non-compound selectors (standard complexes), their coverages are computed in the same way as for standard complexes. The computation of cumulative coverage requires additional effort by matching all previous selectors. To increase efficiency, the coverage is computed once and stored in data structures associated with complexes. Displaying the above coverages can be switched off by setting the *display_selectors_coverage* parameter to "no" in the input file.

3 Representation of compound attributes

This section briefly describes representation of compound attributes in the AQ21 learning system. To make the paper self-contained, Sections 3.1 and 3.2 describe methods for representing standard, discrete, and continuous attributes, respectively. Section 3.3 describes representation of compound attributes.

3.1 Bitstring Representation of Discrete Attributes

Because discrete and continuous attributes are represented differently, these two types of attributes are handled in different ways. Discrete attributes are represented by bitstrings, and continuous attributes are represented by ranges of values [12].

In the bitstring representation, both events and complexes are represented by equal-length binary strings. Each such bitstring is a concatenation of the characteristic vectors of the selector references. The length of a bitstring is thus: $\#D(x1) + ... + \#D(xn) + n$, where $D(xi)$ is domain of the attribute xi, and $\#D$ denotes the cardinality of D. The value n in the formula is added to account for the representation of unknown meta-values. In this representation, each bit indicates the presence (denoted by "1") or absence (denoted "0") of the attribute value corresponding to the bit's position in the string. For example, if the domain of x, D(x), is {0,1,2,3,4}, then value x = 3 is represented by a string $< 000100 >$. Thus, in a representation of an event only one bit is set to "1"for each attribute value in the event. The additional bit at the end of each attribute, a metabit, is set to 1 when meta-value "unknown" is assigned to the attribute [11]. For example, the event $e1 = (color = green)(size =?)$ is represented by the bitstring $< (0100)(0001) >$, assuming that the domain D(size) is small, medium, large. Complexes are represented similarly to events, with one difference - there is no limitation on number of bits set to "1." For example a complex $[color = red\ v\ green]\&[size = small]$ is represented as $< (1100)(1000) >$.

Compound attributes are transformed into basic attributes before learning. Thus, they are represented as concatenation of bitstrings representing constituent attributes. Such representation applies to both events and complexes. For example a compound selector $[head : (red\ v\ green)\&small]$ is equivalent to the complex $[head.color = red\ v\ green]\&[head.size = small]$, and is represented by the bitstring $< (1100)(1000) >$.

3.2 Range Representation of Continuous Attributes

Selectors with continuous attributes are represented in AQ21 by ranges (pairs of real values), in which the first number is the lower bound, and the second number is the upper bound on the values of a given attribute. Both events and complexes are represented this way which means that a selector with continuous attribute can consist only of one range. Such constraint is consistent with human perception and follows the idea of natural induction.

For example, suppose "distance" (in meters) is a continuous attribute, whose domain ranges from 0 to 1000. An event $e1(distance = 37.25)$ would be represented by the pair $(37.25, 37.25)$, in which the lower bound and the upper bound are the same. If an attributional condition is $[distance = 25.3..32.1]$, the program would represent it by the pair $(25.3, 32.1)$ associated with the attribute "distance."

3.3 Representation of Compound Selectors

Compound attributes are converted into groups of constituent attributes before passing them to the AQ learning module. This means that a compound selector is represented as a concatenation of bitstrings and/or ranges used to represent its constituent attributes.

Let a compound attribute "head" consist of basic attributes "size," "shape," and "color," where D(size) is a real value in the range 0..100 in given units, $D(shape) = \{square, triangle, oval\}$, and $D(color) = \{red, green, blue, yellow\}$. According to the description in previous section, "size" is represented by a single range, "shape" by four bits, and "color" by five bits. Thus, the compound attribute head is represented by one range and nine bits. The selector $[head : 12..17.5\ inches\ long \& red\ color]$ is represented by the range $(12, 17.5)$ and the bitstring $< (1000)(11110) >$. Note that the constituent attribute "shape" (the second attribute in the bitstring) is not present which is represented by setting all bits but the metabit to one.

4 Example Application

A simple example is used to illustrate AQ21 learning with compound attributes. Suppose that the problem is to learn a ruleset for discriminating between two types of single-family houses: low-cost and medium-cost. Attributes used to describe the houses are defined in AQ21 in the following form:

```
Attributes {
  location compound {
    distance_to_work continuous
    shopping_center linear {close, relatively_close, far}
    neighborhood nominal {good, bad} }
  number_of_bedrooms linear 4
  number_of_bathrooms linear 4
  master_bedroom compound {
    size linear {small, medium, large}
    balcony nominal {with, without} }
  second_bedroom compound {
    size linear {small, medium, large}
    balcony nominal {with, without} }
  third_bedroom compound {
    size linear {small, medium, large}
    balcony nominal {with, without} }
  yard compound {
    size linear {very_small, small, medium, large, very_large}
    view nominal {great_view, poor_view} }
  garage nominal {yes, no}
  house_type nominal {low-cost, medium-cost} }
```

Suppose that AQ21 is provided with 6 examples of low-cost houses and 7 examples of medium-cost houses. For illustration, the table below presents a subset of the training examples.

```
10, relatively_close, good, 2, 2, medium, with, small, without,
  small, N/A, N/A,   poor_view, yes,        low-cost
10, relatively_close, good, 2, 1, medium, with, small, without,
  small, N/A, N/A,   poor_view, yes,        low-cost
2,  relatively_close, good, 2, 2, medium, with, small, without,
  small, N/A, N/A,   poor_view, yes,         medium-cost
10, close,            good, 2, 2, medium, with, small, without,
  small, N/A, N/A,   poor_view, yes,         medium-cost
20, relatively_close, good, 2, 2, medium, with, small, without,
  small, N/A, N/A,   poor_view, yes,         medium-cost
10, relatively_close, good, 3, 2, medium, with, small, without,
  small, without,  small, poor_view, yes,     medium-cost
```

N/A means here that an attribute is not applicable (e.g., there is no third bedroom). Given the above input, AQ21 generated the following compound attributional rules:

```
[house_type = low-cost]
     # Rule 1
 <-- [number_of_bedrooms=2: 5,5,50%,5,5,50%]
     [master_bedroom: small & without balcony: 3,1,75%,3,0,100%]
         : p=3,n=0,u=3
     # Rule 2
 <-- [location: 6..15 miles distance_to_work &
              relatively_close_shopping_center: 2,1,66%,2,1,66%]
     [number_of_bedrooms= 2: 5,5,50%,2,0,100%]: p=2,n=0,u=2
     # Rule 3
 <-- [location=far_from_shopping_center: 1,1,50%,1,1,50%]
     [number_of_bathrooms=1: 3,2,60%,1,1,50%]
     [garage=no : 3,3,50%,1,0,100%]: p=1,n=0,u=1
```

The first rule above can be paraphrased: "*The house type is low-cost, if number of bedrooms is 2 and the master bedroom is small and without balcony.*" The corresponding rules in a non-compound form are:

```
[house_type = low-cost]
      # Rule 1
  <-- [number_of_bedrooms=2: 5,5,50%,5,5,50%]
      [master_bedroom.size=small: 3,2,60%,3,1,75%]
      [master_bedroom.balcony=without: 3,2,60%,3,0,100%]
         : p=3,n=0,u=3
      # Rule 2
  <-- [distance_to_work=6..15: 2,2,50%,2,2
      [location.shopping_center=relatively_close: 2,3,40%,2,1,66%]
      [number_of_bedrooms=2: 5,5,50%,2,0,100%]: p=2,n=0,u=2
      # Rule 3
  <-- [location.shopping_center=far: 1,1,50%,1,1,50%]
      [number_of_bathrooms=1: 3,2,60%,1,1,50%]
      [garage=no: 3,3,50%,1,0,100%]: p=1,n=0,u=1
```

As one can see, compound rules are simpler and closer to equivalent natural language expressions. Numerical parameters displayed in selectors were described in Section 2.4. In addition, each rule is also annotated by parameters: p-positive coverage, n-negative coverage, and u-unique coverage (the number of training examples covered only by the given rule). More details on these annotations are in [13]. The use of compound attributes is user-controlled by the setting *display_compound_attributes* parameter whose default value is true.

5 Related Research and Summary

As indicated before, compound attributes, introduced in Attributional Calculus, facilitate learning descriptions of objects whose parts are described by different sets of attributes. Learning descriptions with compound attributes is related to learning relational descriptions, which characterize objects in terms of properties of their parts and relations between the parts [7,2,6]. Although compound attributes address only the problem of describing parts of an object in terms of different attributes, and not the relations between the parts, they are sufficient and useful for many application domains. Their advantages are that they are easy to interpret and computationally much simpler to implement than full-fledged relational descriptions. Compound attributes increase descriptions' simplicity, and their understandability because they facilitate expressing knowledge in forms closely related to those used in natural language. Compound attributes are a useful addition to structured attributes in implementing inductive learning. Structured attributes, first implemented in the INDUCE relational learning program, e.g. [5] have hierarchically ordered domains.

This research is also related to the currently very active area of Probabilistic Relational Learning that combines statistical and relational learning (PRMs) e.g. [1,3]. The latter methods concern descriptions of relational data that are in some sense close to descriptions with compound attributes. These methods, however, do not address the problems of natural induction that specifically stresses the comprehensibility and human-orientation of knowledge to be learned.

As shown in the paper, descriptions with compound attributes are simpler than those that do not employ such attributes, and directly correspond to logically equivalent natural language descriptions. The presented method of learning descriptions with compound attributes was implemented in the AQ21 multitask learning, downloadable from www.mli.gmu.edu/software, and described in [13].

6 Acknowledgements

Authors thank Kenneth Kaufman for comments and proofreading of this paper. This research was conducted in the GMU Machine Learning and Inference Laboratory, whose research activities are supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476. The findings and opinions expressed here are those of the authors, and do not necessarily reflect those of the above sponsoring organizations.

References

1. Dey, D. and Sarkar, S. (1996) A Probabilistic Relational Model and Algebra. ACM Transactions on Database Systems (TODS), 21, Issue 3
2. Dietterich, T. G. and Michalski, R. S. (1981) Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods. Artificial Intelligence Journal, Vol. 16, No. 3, 257–294
3. Getoor, L., Friedman, N., Koller, D., Taskar, B. (2001) Probabilistic Models of Relational Structure. International Conference on Machine Learning, ICML'01, Williamstown, MA
4. Kaufman, K. and Michalski, R. S. (2000) An Adjustable Rule Learner for Pattern Discovery Using the AQ Methodology. Journal of Intelligent Information Systems, 14, 199–216
5. Larson, J. and Michalski, R. S. (1977) Inductive Inference of VL Decision Rules," Invited paper for the Workshop in Pattern-Directed Inference Systems, Hawaii, and published in SIGART Newsletter, ACM, No. 63, 38–44.
6. Lavrac, N. and Dzeroski, S. (1994) Inductive Logic Programming: Techniques and Applications, Ellis Horwood
7. Michalski, R. S. (1980) Pattern Recognition as Rule-Guided Inductive Inference. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, 349–361
8. Michalski, R. S. (2004) ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction. Reports of the Machine Learning and Inference Laboratory, MLI 04-2, George Mason University, Fairfax, VA
9. Michalski, R. S. (2004) Generating Alternative Hypotheses in AQ Learning. Reports of the Machine Learning and Inference Laboratory, MLI 04-6, George Mason University, Fairfax, VA
10. Michalski, R. S. and Kaufman, K. (2001) The AQ19 System for Machine Learning and Pattern Discovery: A General Description and User's Guide. Reports of the Machine Learning and Inference Laboratory, MLI 01-2, George Mason University, Fairfax, VA
11. Michalski, R. S. and Wojtusiak, J. (2005) Reasoning with Meta-values in AQ Learning. Reports of the Machine Learning and Inference Laboratory, MLI 05-1, George Mason University, Fairfax, VA
12. Michalski, R.S. and Wojtusiak, J. (2006) Semantic and Syntactic Attribute Types in AQ Learning. Reports of the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA (to appear)
13. Wojtusiak, J. (2004) AQ21 User's Guide. Reports of the Machine Learning and Inference Laboratory, MLI 04-3, George Mason University, Fairfax, VA