Discovering Multi-head Attributional Rules in Large Databases

Cezary Głowiński, Ryszard S. Michalski*

Machine Learning and Inference Laboratory, School of Computational Sciences, George Mason University, Fairfax, VA

*Also with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract: A method for discovering multi-head attributional rules in large databases is presented and illustrated by results from an implemented program. Attributional rules (a.k.a. attributional dependencies) can be viewed as generalizations of standard association rules, because they use more general and expressive conditions than those in the latter ones, and by that can express more concisely inter-attribute relations in a database. Multi-head rules have multiple conditions/statements in their conclusion. The presented method applies AQ learning to create single-head characteristic rules, and then seeks conditions (selectors) that can be transferred to the conclusion part of the rule. Experiments with the program MAR1 (Multi-head Attributional Rules), implementing the developed method, has produced highly encouraging results.

Keywords: attributional rules

1. Introduction

The problem of discovering unknown relations among groups of attributes in a large database is one of the most challenging and practically important problems in the field of data mining. A popular form of expressing such relations in data mining are association rules [1]. In the original formulation, such rules were in the form in which a conclusion consists of only one transaction item (literal). Subsequently, this form was extended to many items in the conclusion [2].

This paper is concerned with discovering more general relations in the data, called attributional rules (or attributional dependencies). An attributional rule is a relation between conjunctions of attributional conditions, expressed as an implicative or equivalence expression in attributional calculus [9]. An elementary form of an attributional condition is a relation between an attribute and a set of

values. A more general form can represent a simple relation between attributes (see Section 2).

The concept of attributional rule, as well as of an association rule, can be viewed as a special case of a more general relational structure, called *parametric association rule* or, briefly, PAR, introduced in [7]. An advantage of using attributional rules in data mining is that they allow the system to express succinctly implicative or equivalence relations among statements involving discrete and/or continuous variables.

Among important problems in data mining are the determination of characteristic rules and rules with low support. Solving them is typically a time consuming task. The method proposed in this paper addresses both problems. It can generate characteristic descriptions as well as attributional dependencies with low support too. A characteristic rule for a class (or concept) is a logic-style description that specifies features that are common to all objects in the class (defined by a specific value of the output attribute). Such rules introduce a approach to characterizing classes of entities, different from statistical methods (e.g., [3]).

In many cases rules with high confidence and low support are very interesting for analysts. Our method differs from other algorithms generating association rules because it allows discover weak patterns with high confidence. Mentioned algorithms discover only strong patterns in practice. Using characteristic rules our algorithm can discover such kind of a relationship easily. The presented method is very efficient. During each iteration a database is scanned only once. Moreover this method does not generate new selectors or numerical intervals but use only these generated during learning of characteristic rules. This speeds up generation of attributional dependencies.

2. Attributional dependencies

This section describes the concept of multihead attributional rule, and defines closely related terms.

2.1. Formal model of attributional dependencies

The basic element used to describe dependencies in data is called *attributional condition* (or simply selector). First time it was introduced in [4] and later in [5]. There are two main types of attributional conditions: nominal and continuous. Attributional condition nominal describes dependency between attribute and a nominal value or a disjunction of values. There are two relations, which can be used inside such condition: equal and not equal. Attributional condition continuous describes dependency between attribute and a value or an interval of values.

The basic form of an attributional condition for attributes with a nominal domain is:

$$[\ \boldsymbol{A}\ rel\ \boldsymbol{V}\ ],$$

where *rel* is a relation symbol from the set {=,!=}, ***A*** is an attribute, ***V*** is a value or an disjunction of values.

The basic forms of an attributional condition for attributes with a continuous (or numerical) domain are:

$$[\ \boldsymbol{A}\ rel\ \boldsymbol{V}\ ]$$

or

$$[\ \boldsymbol{A} = \boldsymbol{I}\ ],$$

where *rel* is a relation symbol from the set {=,!=,<,>,<=,>=}, ***A*** is an attribute, ***V*** is a value and ***I*** is a range describing interval between two values.

Attributional condition [**A** *rel* **V**] is satisfied by the example (or is true) if for this example the value of attribute ***A*** is in relation *rel* to ***V***.

A conjunction of attributional conditions is called a complex. We say that a complex is satisfied by an example if all attributional conditions in the complex are satisfied.

An attributional dependency is a rule in implicate form $X=>Y$ or in equivalence form $X<=>Y$, where X and Y are complexes. In other words, an attributional dependency describes relationship between conjunctions of attributional conditions. The rule is satisfied by an example if both a condition and a conclusion, which are complexes, are satisfied.

To select most significant dependencies we use coefficients well known in data mining and statistics like *support*, *confidence* and *lift*. We also introduce a coefficient called *concept coverage*, shortly *cover*.

2.2. Definitions and interpretation of coefficients

Support

Support is the base coefficient used to make a rule assessment. It says how strong is a dependency. We say that support is equal *s* if *s%* of examples from database satisfy dependency. It's mean that if support is high then dependency is satisfied by many examples from a database. In particular we define support for a complex and a rule. A value of this coefficient belongs to the interval [0,1].

Support for a complex C_i is defined as follows:

$$supp(C_i) = \#\ examples\ satisfying\ C_i\ /\ N,$$

where N is a number of all examples in a database.

Support for a rule *R* is defined as follows:

$$supp(R) = supp(X=>Y) = \#\ examples\ satisfying\ X\ and\ Y\ /\ N.$$

Confidence

Confidence describes the probability of dependency. We say that confidence is equal *c* if *c%* of examples that satisfy condition also satisfy conclusion of a rule. High confidence means that if condition is true then is high probability that also conclusion is true. A value of this coefficient belongs to the interval [0,1].

Confidence for a rule *R* is defined as follows:

$$conf(R) = conf(X=>Y) = supp(X\ and\ Y) / supp(X).$$

Lift

Lift describes the interestingness of the rule. It says how strong is the association between a condition and a conclusion of the rule. Based on statistical independence we can say that if lift is equal 1 then the condition and the conclusion are independent. If lift is greater than 1 then the condition is associated with the conclusion. If lift of a rule is between 0 and 1 then a condition is negatively associated with a conclusion.

Lift for a rule *R* is defined as follows:

$$lift(R) = lift(X=>Y) = N\ conf(X=>Y) / supp(Y) = N\ supp(X\ and\ Y) / supp(X)\ supp(Y).$$

This coefficient has only informative character and isn't used to prune rules during a mining process.

Concept coverage

Concept coverage describes the importance of dependency. We say that concept coverage is equal *c* if *c%* of examples that satisfy conclusion also satisfy condition of a rule. High concept coverage means that dependency highly covers chosen concept. A value of this coefficient belongs to the interval [0,1].

Concept coverage for a rule *R* is defined as follows:

$$cover(R) = cover(X=>Y) = supp(X\ and\ Y) / supp(Y).$$

If *cover(R)* is equal 1 we can say that the rule *R* covers whole a concept *Y*. If *cover(R)* is equal to 0 then we can say that the rule *R* doesn't cover a concept *Y*. The rule *R* with higher *cover* is more general. The advantage of concept coverage is that it can be used to rules pruning during mining process.

Equivalence tolerance

The MAR1 algorithm can generate attributional dependencies in the form of implicate rules and equivalence rules. Equivalence rule says that two complexes are equivalent. Theoretically this situation comes when confidence is equal 1 and concept coverage is equal 1. Introducing of equivalence tolerance coefficient allow us to generate approximate (or fuzzy) equivalence rules. Using this parameter we can treat all dependencies which meet the following condition as equivalence:

conf(R) >= (1-equivalence tolerance) and cover(R) >= (1-equivalence tolerance),

where equivalence tolerance is in [0,1]. In practice, the value of equivalence tolerance parameter should be low. If the value of equivalence tolerance parameter is equal 0 we get real equivalence rules.

2.3. Methodology and theoretical framework

The methodology of discovering attributional dependencies assume that the process is consists of two steps. In the first step we learn characteristic rules (description) for given attribute or set of attributes using machine learning program, i.e. AQ or INLEN. In the second step, we generate attributional dependencies based on learned rules.

Let's assume we have characteristic rule in the following form:

$$a_1 \,\&\, a_2 \,\&\, a_3 \,\&\, a_4 \,\&\, ... \&\, a_i \,\&\, ... \,\&\, a_n \Rightarrow b_1$$

Based on inferential theory of learning [8] if we want to move selector *ai* from the condition to the conclusion we have to check support of two relationships:

$$R_1: \; a_1 \,\&\, a_2 \,\&\, a_3 \,\&\, a_4 \,\&\, ... \,\&\, a_n \Rightarrow a_i$$

and

$$R_2: \; a_1 \,\&\, a_2 \,\&\, a_3 \,\&\, a_4 \,\&\, ... \,\&\, a_n \Rightarrow \textbf{not}\ a_i$$

The support of the first rule is denoted as $supp(R_1)$ and the second rule is denoted as $supp(R_2)$. If the $supp(R_1)$ is high in compare to $supp(R_2)$ then we can suppose that it is possible to move attributional condition (selector) a_i to the conclusion. In other case we can't do it. To describe difference between $supp(R_1)$ and $supp(R_2)$ we compute coefficient *trans*. This coefficient expresses relative relation between supports of both rules in percents and is defined as follows:

$$trans(a_i) = supp(R_1) \,/\, (supp(R_1) + supp(R_2))$$

The coefficient *trans* belongs to the interval [0,1] from definition.

To decide if we can move selector from a condition to a conclusion we use input threshold parameter called *mintrans*. If $trans(a_i) > mintrans$ we can generate new rule in the following form:

$$a_1 \,\&\, a_2 \,\&\, a_3 \,\&\, a_4 \,\&\, ... \,\&\, a_n => b_1 \,\&\, a_i$$

Usually the value of *mintrans* parameter should be close to 1, then the process of new rules generation is most efficient.

This methodology allows discovering attributional dependencies for both nominal and continuous values.

2.4. Algorithm MAR1

In previous section we described the methodology of attributional dependencies generation. Now, we will show the new algorithm, called MAR1 (*Multi-head Attributional Rules*), which implements presented ideas in practice. This algorithm consists of three phases:

- reading and analyzing characteristic rules,
- selecting rules,
- generating attributional dependencies.

In the first phase algorithm reads characteristic rules (procedure `ParseRules()`) form a ruleset. Next, we generate simple rules with single conjunction (procedure `AnalizeRules()`) from complex rules (which have alternative of conjunctions in a condition part). In second phase we prune these characteristic rules, which have support less than *minsup* (procedure `SelectRules()`). The most interesting is third phase, where the algorithm generates attributional dependencies (procedure `Generate Dependencies()`).

```
algorithm MAR1(minsup, minconf, mincover, mintrans)
begin
  ruleset=0
  ParseRules(ruleset);
  AnalizeRules(ruleset);
  SelectRules(ruleset, minsup);
  GenerateDependencies(ruleset, minconf, mincover, mintrans);
end
```

Fig. 1 Pseudo-code of the MAR1 algorithm

Procedure for generating attributional dependencies takes a ruleset and a database. At the beginning this procedure scans a database and check if selectors are satisfied by examples. With each selector in a rule there is associated vector of flags. The value of the flag *i* is 1 if an attributional condition is satisfied by *i*th example in a database. Otherwise the value is 0. The advantage of this solution is that we scan database only once during iteration. In the next step, procedure computes the coefficient *trans* for all selectors in the condition of the rule. If `trans(s)` is greater than *mintrans* then procedure may generate a new rule. We transfer selected attributional condition from the condition to the conclusion. Then we compute confidence and cover coefficients for new rule. If both are greater than minimal threshold values we add the rule to new ruleset. If new ruleset is not empty we continue starting next iteration.

```
procedure GenerateDependencies(ruleset, minconf, mincover,
mintrans)
while ruleset<>0
begin
  newruleset=0;
  ScanDatabase(ruleset);
  for all r in ruleset
  begin
    for all selectors s in r
    begin
      Compute(trans(s));
      if trans(s)>=mintrans then
        begin
          newr=Transfer(r,s);
          Compute(conf(newr) and cover(newr));
          if conf(r)>=minconf and cover(r)>=mincover then
            newruleset=newruleset+newr;
        end
    end
  end
  ruleset=newruleset;
end
```

Fig. 2 Procedure for generating dependencies

3. An example and other experiments

In this paper we present results from our algorithm for database zoo. It contains data about animals (17 attributes and 108 examples). We set input parameters as follow: number of iterations = 3, trans threshold = 0.9, support threshold = 0.2, confidence threshold = 0.2, cover threshold = 0.8 and equivalence

tolerance = 0.05. Using INLEN we found 8 characteristic rules for attribute airborne. According to threshold values MAR1 selected 2 rules:

```
IF [type != 2 or 6] and [feathers = 0] THEN [airborne = 0]     {
supp=0.72   conf=0.90   cover=0.95   lift=1.18 }
IF [legs = 2 or 6] and [catsize = 0] and [type = 1 or 2 or 6] and
[breathes = 1] and [fins = 0] THEN [airborne = 1]     { supp=0.21
conf=0.84   cover=0.88   lift=3.54 }
Rules #2
```

Based on above characteristic rules MAR1 generated 4 rules in first iteration, 6 rules in second iteration and 6 rules in third iteration. We present rules obtained in second:

```
Iteration=2
IF [legs = 2 or 6] and [catsize = 0] and [fins = 0] THEN [airborne =
1] and [type = 1 or 2 or 6] and [breathes = 1]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
IF [legs = 2 or 6] and [catsize = 0] and [breathes = 1] THEN
[airborne = 1] and [type = 1 or 2 or 6] and [fins = 0]     {
supp=0.21   conf=0.84   cover=0.88   lift=3.54   trans=1.00 }
IF [legs = 2 or 6] and [catsize = 0] and [fins = 0] THEN [airborne =
1] and [breathes = 1] and [type = 1 or 2 or 6]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
IF [legs = 2 or 6] and [catsize = 0] and [type = 1 or 2 or 6] THEN
[airborne = 1] and [breathes = 1] and [fins = 0]     { supp=0.21
conf=0.84   cover=0.88   lift=3.54   trans=1.00 }
IF [legs = 2 or 6] and [catsize = 0] and [breathes = 1] THEN
[airborne = 1] and [fins = 0] and [type = 1 or 2 or 6]     {
supp=0.21   conf=0.84   cover=0.88   lift=3.54   trans=1.00 }
IF [legs = 2 or 6] and [catsize = 0] and [type = 1 or 2 or 6] THEN
[airborne = 1] and [fins = 0] and [breathes = 1]     { supp=0.21
conf=0.84   cover=0.88   lift=3.54   trans=1.00 }
Rules #6
```

and third iteration:

```
Iteration=3
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and [type =
1 or 2 or 6] and [breathes = 1] and [fins = 0]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=1.00 }
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and [type =
1 or 2 or 6] and [fins = 0] and [breathes = 1]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and
[breathes = 1] and [type = 1 or 2 or 6] and [fins = 0]     {
supp=0.21   conf=0.78   cover=0.88   lift=3.27   trans=1.00 }
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and
[breathes = 1] and [fins = 0] and [type = 1 or 2 or 6]     {
supp=0.21   conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and [fins =
0] and [type = 1 or 2 or 6] and [breathes = 1]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
```

```
IF [legs = 2 or 6] and [catsize = 0] THEN [airborne = 1] and [fins =
0] and [breathes = 1] and [type = 1 or 2 or 6]     { supp=0.21
conf=0.78   cover=0.88   lift=3.27   trans=0.92 }
Rules #6
```

Results for mushroom database. It contains data about mushrooms (23 attributes and 8124 examples). We set input parameters as follow: number of iterations = 3, trans threshold = 0.9, support threshold = 0.2, confidence threshold = 0.9, cover threshold = 0.9 and equivalence tolerance = 0.05. Using INLEN we found 2 characteristic rules for attribute A0. Below we show example rule generated by MAR1:

```
IF [odor = a or l or n] and [cap-shape = b or x or f or s] and [cap-
surface != g] and [cap-color = n or y or w or g or e] and [gill-
attachment = f] and [gill-spacing = c or w] and [gill-color = k or n
or h or g or p or u or w] and [stalk-root = b or c or e or r] and
[stalk-surface-above-ring = f or s] and [stalk-surface-below-ring !=
k] and [stalk-color-above-ring = g or p or w] and [stalk-color-below-
ring = g or p or w] and [veil-type = p] and [veil-color = w] and
[ring-number = o] and [ring-type = e or p] and [spore-print-color = k
or n or u] and [population != c] and [habitat != l or w] THEN
[mushroom = e]
```

from original characteristic rule:

```
IF [odor = a or l or n] and [gill-attachment = f] and [gill-spacing =
c or w] and [gill-color = k or n or h or g or p or u or w] and
[stalk-root = b or c or e or r] and [stalk-surface-above-ring = f or
s] and [stalk-surface-below-ring != k] and [stalk-color-above-ring =
g or p or w] and [stalk-color-below-ring = g or p or w] and [veil-
type = p] and [veil-color = w] and [ring-number = o] and [ring-type =
e or p] and [spore-print-color = k or n or u] and [population != c]
and [habitat != l or w] THEN [mushroom = e] and [cap-shape = b or x
or f or s] and [cap-surface != g] and [cap-color = n or y or w or g
or e] { supp=0.42   conf=1.00   cover=0.85   lift=2.04   trans=1.00 }
```

We carried out several experiments with MAR1 using benchmark databases from UCI Repository. Then we compared our results with result from Apriori algorithm [2] for finding association rules. In most cases additional rules were discovered that were not discovered by Apriori.

Comparing a rulesets obtained from MAR1 and Apriori algorithms we can say that Apriori algorithm usually generates more rules. However, rules generated by MAR1 are more interesting because most of them have high confidence and high cover (even if support is low). Apriori especially prefers rules with high support. To generate rules with particular attribute in a conclusion part users need additional rules filtering in Apriori. Summarizing, the most general

difference between algorithms is that Apriori focuses on rule generation directly from data, while MAR1 generates multi-head rules from other single-head rules.

4. Related works, conclusions and future research

The concept of an attributional dependency is a generalization of an association rule. An association rule is an implication of the form $X=>Y$, where X and Y are sets of items [1], where items are binary attributes. Our definition of attributional dependency says that a condition and a conclusion are conjunction of attributional conditions. Moreover we can formulae dependencies in either implicative or equivalence form. Known algorithms for discovery rules use only data to formulae rules. In this paper we presented new methodology to formulae rules from other rules especially we propose use of characteristic rules because they are most informative. Based on them and a database we discover attributional dependencies. In practice miners are interested in finding dependencies consist of selected feature because discovering all rules in database is very complex task. The advantage of our method is this that we generate only most characteristic dependencies for chosen attributes. Moreover, there are many interesting and useful dependencies with low support but with high confidence. Known algorithms need a lot of time to discover association rules with low support. Proposed methodology allows discover either strong or weak relationships without additional computation. To find most interesting association rules there are used statistical measures based on rules occurrence frequency in the database. We also proposed coefficient (called concept coverage) as one of measure of interestingness of dependency. This coefficient is used to prune rules during process of generation. Coverage says how a rule covers a concept interested for us.

This research was completed but we see some opportunities for future work. In our method may be introduced better selection procedure, which will help to select more interesting rules. Currently select procedure is based on coefficients (like support, confidence, cover, lift and trans). It is also possible to select rules according to the content of the condition and the conclusion. It means that user can describe what attribute or set of attributes should be in the head and/or body of the rule interesting for him.

Moreover, MAR1 algorithm can be implemented as new operator in KGL language (*Knowledge Generation Language*). In the future MAR1 module can be invoked by KGL using operator ADGEN with proper parameters. Implementing this feature will require a modification of the KGL parser.

Acknowledgements

This research has been conducted in Machine Learning and Inference Laboratory at George Mason University. Cezary Głowiński has been supported by Kosciuszko Foundation in New York, in part by the National Science Foundation

Grant IIS 9906858, and in part by the UMBC/MPO/Lucite #32 grant. The authors especially thank Guido Cervone for his great help during our research.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules between Sets of Items in Large Databases, *Proceedings of VLBD-93*, 1993
2. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules, *Proceedings of VLBD-94*, 1994
3. Bayardo R.J., Agrawal R.: Mining the Most Interesting Rules, *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999
4. Michalski R.S.: AQVAL/1-Computer Implementation of a Variable-Valued Logic System VL1, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, 1973
5. Michalski R.S.: Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning, chapter in *Computer Science and Multiple-Valued Logic Theory and Applications* Rine D.C. (ed.), North-Holland Publishing, 1975
6. Michalski R.S.: A Theory and Methodology of Inductive Learning, *Artificial Intelligence*, 1983
7. Michalski R.S.: Toward a Unified Theory of Learning: Multistrategy Task-adaptive Learning, in *Readings in Knowledge Acquisition and Learning* Buchanan B.G., Wilkins D.C., 1991
8. Michalski R.S.: Inferential Theory of Learning: Developing Foundations for Multistrategy Learning, chapter in *Machine Learning A Multistrategy Approach Volume IV* Michalski R.S., Tecuci G. (eds.), Morgan Kaufman Publishers, 1994
9. Michalski, R.S.: Natural Induction and Concept Learning: The AQ Methodology and Its Application to Machine Learning and Knowledge Mining, 2001, to appear.