ATTRIBUTIONAL CALCULUS

A Logic and Representation Language for Natural Induction

Ryszard S. Michalski

April, 2004

MLI 04-2
P 04-2

ATTRIBUTIONAL CALCULUS: A LOGIC AND REPRESENTATION LANGUAGE FOR NATURAL INDUCTION

Ryszard S. Michalski

Machine Learning and Inference Laboratory, George Mason University,
Fairfax, VA 22030-4444
and

Institute of Computer Science, Polish Academy of Sciences, Warsaw

michalski@mli.gmu.edu
http://www.mli.gmu.edu

Abstract

Attributional calculus ($\mathcal{AC}$) is a typed logic system that combines elements of propositional logic, predicate calculus, and multiple-valued logic for the purpose of *natural induction*. By natural induction is meant a form of inductive learning that generates hypotheses in human-oriented forms, that is, forms that appear natural to people, and are easy to understand and relate to human knowledge. To serve this goal, $\mathcal{AC}$ includes non-conventional logic operators and forms that can make logic expressions simpler and more closely related to the equivalent natural language descriptions. $\mathcal{AC}$ has two forms, *basic* and *extended,* each of which can be *bare* or *annotated*. The extended form adds more operators to the basic form, and the annotated form includes parameters characterizing statistical properties of bare expressions. $\mathcal{AC}$ has two interpretation schemas, *strict* and *flexible*. The strict schema interprets $\mathcal{AC}$ expressions as true-false valued, and the flexible schema as continuously-valued. Conventional decision rules, association rules, decision trees, and *n*-of-*m* rules all can be viewed as special cases of attributional rules. Attributional rules can be directly translated to natural language, and visualized using *concept association graphs* and *general logic diagrams*. $\mathcal{AC}$ stems from Variable-Valued Logic 1 (VL_1), and is intended to serve as a concept description language in advanced AQ inductive learning programs. To provide a motivation and background for $\mathcal{AC}$ the first part of the paper presents basic ideas and assumptions underlying concept learning.

Keywords: Inductive inference, machine learning, natural induction, data mining and knowledge discovery, knowledge mining, propositional calculus, predicate logic, many-valued logic, attributional calculus

Acknowledgments

The author thanks Ken Kaufman for many valuable comments and suggestions that helped to improve an earlier version of the paper, in particular regarding the section on censored and preconditioned rules. Thanks go also to John Ashley, Chien-Chih Lin and Jim Jones, students in the Computational Learning and Inference class, for helping to debug the paper.

This research was conducted in the Machine Learning and Inference Laboratory of George Mason University whose research activities are supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant.

1 MOTIVATION

The explosive growth of databases and information systems world-wide has created a critical need for automating processes of minting useful knowledge from available data, and of updating it with new data. Computer systems capable of performing such tasks in an effective manner will make a vital contribution to almost all areas of human endeavor, including sciences, engineering, business, economics, medicine, bioinformatics, agriculture, management, archaeology, and defense. This is so because all these areas have been accumulating computer-stored datasets and have a need to use them for creating or improving existing knowledge.

The fast growing field of data mining is concerned with deriving useful patterns from large data volumes, typically without engaging much background knowledge and without conducting sophisticated symbolic reasoning. Many data mining methods are now available, such as decision trees, decision rules, Bayesian nets, support vector machines, nearest neighbors, artificial neural nets, etc.

There is, however, a class of problems complementary to those considered in data mining, namely problems concerned with determining useful knowledge from impoverished data---scarce, uncertain, inconsistent, imprecise, indirectly relevant, and/or erroneous data, which in addition may be distributed over space and/or time. The ability to derive useful knowledge from such impoverished data is vastly important for a wide range of applications. Such applications include early disease diagnosis, illegal activity detection, computer system intrusion detection, security breach recognition, fraud detection, terrorist attack prevention, economic decision making, industrial espionage, military decision making, and many others.

When extracting useful patterns from large volumes of data, the main concern is the algorithm's efficiency. When deriving useful knowledge from impoverished data, the main issues are how to engage background knowledge about the problem in order to compensate for the data limitations. In this case, the system has to be able to determine problem-relevant knowledge, and to use it in reasoning about the data.

To illustrate the trade-off between background knowledge and input data in solving problems, consider two questions: 1) approximately how many steps you need to walk from your home to buy bread during daytime, and 2) how many ants live in your neighborhood. To answer the first question, one usually does not need to collect any new data, because one typically knows the location of the nearest food shop, and can estimate the number of steps by simple reasoning. Answering the second question, however, would likely require collecting data about ants in the neighborhood, because one normally does not have much knowledge about such a topic.

Clearly, the ability to use prior knowledge in analyzing limited data is crucial for many applications. Without such knowledge, a system can only determine regularities among attributes and relations already present in the data. It cannot create richer knowledge involving concepts not present in the data. Yet, most of the existing machine learning and data mining methods operate solely on data; without engaging much background knowledge.

To be able to handle both types of problems—deriving useful knowledge from very large volumes of data as well as from impoverished data—one needs systems capable of efficient reasoning about data using background knowledge. The development of such systems requires an integration of methods of machine learning, data mining, and statistics with knowledge representation and symbolic reasoning.

Regardless of whether knowledge is being derived from large volumes of data or from impoverished data, a common issue for both types of problems is the understandability of the knowledge created. In some applications this issue is of minor significance. For example, it is not important for a computer user to understand how the computer recognizes human faces, as long as it does so reliably. In a vast range of other applications, however, computer-generated knowledge must not only be accurate, but also understandable, if it is to be used. A doctor will not make an important medical decision on the basis of a computer's advice if he or she does not understand it. Similarly, a general will not start a battle just because computer said so, but will want to understand the reasons for this advice.

Clearly, humans, especially experts, are very reluctant to employ any machine-generated knowledge blindly, even if that knowledge has been shown to be effective in prior experiments. Although there has been widespread agreement on this issue for a long time, most of the current research in machine learning and data mining has focused primarily on predictive accuracy as the main measure of the system's performance. For example, it is rare to find an article in a machine learning or data mining journal that reports not only the accuracy of knowledge generated from data, but also the actual knowledge generated, and an evaluation of its comprehensibility. In short, the issue of understandability of computer-generated knowledge is an under-investigated problem that needs more attention.

To address the issues raised above, that is, to develop a method for generating understandable knowledge from data, large or small, one needs a knowledge representation and inference system that satisfies three requirements: (1) it represents knowledge in forms easy to understand and interpret, (2) it can reason with both data and prior knowledge, (3) it can be efficiently implemented for conducting both inductive inference (to generate knowledge) and deductive inference (to test and apply knowledge). The attributional calculus, presented in this paper, aims at satisfying these conditions for a limited but important class of problems.

2 THE CONCEPT OF NATURAL INDUCTION

To explicitly address problems in developing inductive inference systems with the capability to create understandable knowledge, we introduced the concept of *natural induction*, defined as an inductive learning process that strives to generate hypotheses that are both accurate and natural for people, and that are easy to interpret and explain (Michalski, 1999). What form of knowledge is "natural" depends, of course, on the person's prior training and experience. A mathematician's proof of a theorem, a doctor's description of a patient's disease, or an engineer's design of an integrated circuit may be "natural" to them, but not to a person untrained in their areas. It is, therefore, inherently impossible to develop a universal, prior knowledge-independent measure of "naturalness" of knowledge forms and knowledge understandability.

One can make, however, a general assumption that "natural" knowledge representations are akin to those in which people typically represent knowledge. If knowledge concerns a specific domain, then natural forms are those used by experts in this domain. If it is general-purpose knowledge, then natural forms are those widely used by people, such as natural language descriptions and visual forms, such as drawings and pictures.

Clearly, natural language is a foundation for all human knowledge representation, knowledge reuse, and exchange. Although different areas of science have developed special formalisms supplementing natural language, for example, mathematical and logical symbols, natural language is the glue that bonds all forms together. In addition to natural language descriptions, visual representations are widely used, as they tend to be highly informative and easy to comprehend, as reflected by the popular adage "a picture is worth a thousand words" (in which "thousand" is not be taken literally, but as an indicator of a large number). It is thus not unreasonable to desire that new knowledge created by smart computers will ultimately appear in the forms used by humans since the beginning of writing over five thousand years ago, such as texts, images, documents, publications, and books, and that its authors will be teams of cooperating computer programs and people, or perhaps just computer programs.

At this early stage of research, we assume that the best representations for effectively implementing natural induction are those whose expressions are syntactically and semantically close to logically equivalent natural language descriptions, and are presentable in easy-to-interpret graphical forms.

3 ISSUES IN IMPLEMENTING NATURAL INDUCTION

As implied above, the basic issue in implementing natural induction is the choice of the representation system for expressing generated hypotheses. A suitable representation system needs to balance trade-offs between its expressive power, its efficiency for implementing automated induction, and its susceptibility to overfitting. A highly expressive representation system will have many forms and use many operators. The more operators and forms the system contains, however, the exponentially larger is the search space, and thus the more complex is the process of inducing descriptions in this space. If the space of possible representations is large, the inductive process is also more prone to data overfitting, that is, to creating hypotheses that fit the training data well, but poorly predict new facts. The idea of natural induction traces its origin to the *postulate of comprehensibility* formulated in (Michalski, 1983, page 94):

> *The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single "chunks" of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.*

As mentioned above, the issue of comprehensibility of computer generated knowledge has not been given sufficient attention in machine learning and data mining. Among scant efforts in this area are a workshop "Machine Learning and Comprehensibility," organized at the 1995 International Joint Conference on Artificial Intelligence (IJCAI '95, Workshop 22) and Pazzani, Mani and Shankle (1997). One of the major reasons for the slow progress is the difficulty of determining an adequate measure of description comprehensibility.

To develop a practical and simple system for natural induction, we sought a formal representation language that has a relatively high expressive power, but is easy to interpret in natural language, and simple to implement in a computer program. Our efforts to satisfy these criteria have led us to the development of *attributional calculus*, a logic and representation system that combines elements of propositional logic, predicate logic, and multiple-valued logic. It is based on our earlier work on variable-valued logic (e.g., Michalski, 1972). Conventional decision rules used in rule learning programs, e.g. CN2 (Clark and Niblett, 1989), or RIPPER (Cohen, 1995) are special cases of attributional rules, and so are *n*-of-*m* rules (Craven and Shavlik, 1996).

As a description language, attributional calculus can be used to represent any set of entities and mappings between them in terms of multi-type, discrete or continuous attributes. The resulting descriptions are compact, and closely correspond to equivalent natural language descriptions. As a logic, attributional calculus facilitates both inductive inference (hypothesis generation) and deductive inference (hypothesis testing and application). It can thus serve as both a simple knowledge representation for inductive learning, and as a system for reasoning about entities described by attributes.

The above features make attributional calculus a useful tool for implementing natural induction, with applications to machine learning, data mining and knowledge discovery. Because it facilitates both automated and manual knowledge representation, it can also be used for a simple form of *knowledge mining*. By this, we mean the process of creating human-oriented knowledge from data (large *or* small) that utilizes non-trivial amounts of background knowledge (BK), in contrast to data mining, which is typically concerned with discovering patterns in large databases without the use of much background knowledge. In knowledge mining, BK can be general common-sense knowledge, as well as specific domain knowledge. BK may be hand-crafted into the system, or created through a previous knowledge mining processes. BK can be in many different forms, but must be amenable to computer manipulation and inference. Attributional calculus is not meant to be a general purpose knowledge representation, but rather a system that allows one to represent statements about objects describable in terms of attributes and their relationships.

As mentioned earlier, in addition to natural language descriptions, knowledge representations utilizing visual forms are also natural to people. Such forms include graphs, diagrams, tables, histograms, networks, maps, flow-charts, schemas, sketches, pictures, 2-D and 3-D models, drawings, images and videos. Therefore, research on natural induction also includes the development of methods for generating and presenting knowledge in visual forms. To this end, we have developed two methods for visualizing attributional calculus expressions, *generalized logic diagrams* (Michalski, 1978a; Wnek, 1995) and *concept association graphs* (Kaufman and Michalski, 2003).

4 TRANSLATION-BASED VS. PROCESS-BASED APPROACHES

One may be tempted to propose an approach to natural induction in which inductive inference is conducted in a representation system alien to human cognition but convenient computationally, and require only that its results be translated to “natural”, human-oriented forms. We call such an approach *translation-based*. The translation-based approach appears attractive because it can be used with any inductive learning system, as it separates hypothesis generation from hypothesis representation. An example of this approach is work by Craven and Shavlik (1996; 1999) on the

TREPAN system for creating decision trees approximating neural networks. The trees can use standard attribute-relation-value tests, or *m*-of-*n* tests (the latter are a special case of attributional expressions with counting attributes—see Section 6).

While this approach can be useful in some situations, it has intrinsic limitations. These limitations are due to the fact that the representation language used for conducting inductive inference, sometimes called a *representational bias*, has a direct influence on the kind of generalizations a system will be able to generate, and then on the selection from among a plethora of possible generalizations that can usually be induced from any non-trivial dataset.

By its very nature, inductive inference is an under-constrained problem; thus the same data may beget many different generalizations consistent with it. The central issue is which generalization to choose from a large space of potential hypotheses. This problem is amplified by the fact that if the induction system uses inadequate representation for a given problem, then regardless of how large the hypothesis space and how robust the induction algorithm it employs, it will be unable to generate a satisfactory generalization. Although the above argument may appear similar to the Sapir-Whorf hypothesis that the language we speak strongly influences our thinking (Whorf, 1956), it has additional significance in the case of inductive generalization, due to the under-constrained nature of such inference.

Specifically, the generation of inductive hypotheses from data typically involves many steps at which intermediate hypotheses (generalizations) are created and evaluated. At each of these steps, an inductive reasoner must make a decision as to which of the alternative intermediate generalizations to explore further, and which to discard (otherwise, the task may become computationally prohibitive). The same generalization expressed in one representation language may score high on a given criterion of hypothesis quality (e.g., a measure of simplicity and/or comprehensibility), but score low when expressed in another language. Consequently, the process of generalization, and the selection of intermediate and the final inductive hypothesis depends on the representation language employed.

This argument has been convincingly supported by simple experiments reported in (Wnek and Michalski, 1992), in which methods that use opaque knowledge representations, such as artificial neural nets and classifiers used by a genetic algorithm were unable to correctly learn target concepts, even when the training set contained *all possible positive* concept examples. In these experiments, the target concepts had very simple natural language representations.

Consequently, an approach based on a *post factum* translation of “alien” representations to logically equivalent (or approximate) natural forms is fundamentally limited from the viewpoint of natural induction. To overcome this limitation, appropriate cognitive constraints must be imposed on the inductive inference process itself. In other words, such a process should be carried out from the outset in a cognitively-oriented representation system. This is a *process-based* approach to natural induction, which has motivated the development of attributional calculus. The Appendix presents an example showing that depending on the representation language, different inductive hypotheses are chosen among alternatives.

5 ATTRIBUTIONAL CALCULUS VS. OTHER LOGICS

Before defining Attributional Calculus ($\mathcal{AC}$) formally, let us characterize it in general, and relate it to other logic systems. Such an introduction will help the reader to quickly understand the motivation behind $\mathcal{AC}$, and its novel features. Attributional Calculus has two forms, *basic* and *extended*, each of which can be *bare* or *annotated*. The combination of the forms implemented in a learning program indicates its learning capabilities.

The basic form can be viewed as a propositional logic in which propositional letters (standing for sentences or their negations) are replaced by *basic attributional conditions*. Such conditions are in the form (*attribute relation valueset*). The extended form adds to the basic form more complex conditions, called *extended conditions*, which involve relations among attributes, *compound* and *derived* attributes, quantified expressions using *counting attributes*, and the *exception* and *provided-that* operators. Attributional conditions, basic and extended, directly correspond to simple natural language sentences describing an entity or a set of entities in terms of one or a few attributes.

In attribute-based learning, each example is described by the same attributes. This limitation is overcome in attributional calculus by compound attributes that consist of attributes that apply only to a part of the entity. Derived attributes are functions of original attributes, and are used to express relationships that require lengthy and complex descriptions if expressed in terms of original attributes. Derived attributes include *counting attributes*, *max*, *min*, *avg*, and function references (which refer to attributes defined by expressions). The counting attributes represent the number of attributional expressions in a given set, and facilitate the construction of a wide range of quantified expressions. By using counting attributes, symmetric logic functions, which tend to have very long descriptions, can be greatly simplified (Michalski, 1969). Function references are links to arbitrary expressions or procedures that compute the function defining the derived attribute. The "exception" operator is used to express descriptions with exceptions, which block the rule if the exception holds. A standard use of such an operator is in *censored rules* that consist of a conjunctive premise and an *exception clause.* The "provided-that" operator is used to define preconditions for applying attributional rules.

The annotated form adds *merit parameters* to bare (basic or extended) attributional expressions. These parameters provide statistical information about the expressions. Such parameters include *coverage* (*positive, negative*, *unique*, and *flexible*), *support*, *confidence*, and *accuracy gain*.

Attributional calculus expressions can be evaluated using two different interpretation schemas, *strict* and *flexible*. The strict schema treats attributional sentences (basic or extended) as binary logic expressions, evaluating them either to truth or falsity. The flexible interpretation schema treats attributional sentences as multi-valued logic expressions. Matching such expressions with observational data produces a degree of match ("truth") that can be continuous or discrete. Thus, $\mathcal{AC}$ becomes a multi-valued logic by a semantic extension, that is, by the way its sentences are interpreted.

$\mathcal{AC}$ can also be extended syntactically to a multi-valued logic system, as was done in its predecessor, VL1, by introducing constants and redefining its operators (Michalski, 1972). This is not done in $\mathcal{AC}$ because expressions in such a system would cease to correspond directly to equivalent natural language sentences. Instead, $\mathcal{AC}$ preserves the spirit of natural language in which the same sentence can be given different degrees of truth depending on the context in which it is evaluated. For example, the sentence "this ant is big" may be true in the context of other ants, but false in the context of rabbits. In the following sections, we will first describe $\mathcal{AC}$ with a strict interpretation, and then discuss various methods of flexible interpretation.

To provide an easy insight into $\mathcal{AC}$, let us briefly discuss its relationship to three well-known logic systems: propositional logic, predicate logic, and multiple-valued logic. Propositional logic uses literals (single propositional letters or their negations) to represent declarative natural language sentences (i.e., sentences that can be evaluated to true or false). More complex expressions are constructed by combining literals by logical connectives, "and", "or", "not", "exclusive-or" and "equivalence" (Kozy, 1974). Because literals stand for entire natural language declarative sentences, they do not convey information about the structure of the sentences they represent. If there is a relationship between components of different sentences, the information about it is lost when the sentences are abstracted into literals.

From the viewpoint of natural induction, the advantages of propositional logic are that its sentences are easy to interpret, and its computer implementation can be very efficient. Its expressive power, however, is too limited for many applications. The attributional conditions introduced in $\mathcal{AC}$ significantly extend the representational power of literals in propositional logic.

Predicate logic (first order and higher order) extends propositional logic by adding to it the ability to represent structure of sentences, and to quantify its expressions (e.g., Carney, 1970; Kozy, 1974; Copi, 1982). The structure and the components of the sentences are represented by multi-argument predicates and terms. In first order predicate logic, an expression can be quantified by an existential quantifier that postulates the existence of at least one entity in the universe of discourse for which the expression is true, or by a universal quantifier that postulates that the expression is true for all entities in the universe. In higher-order logics, the quantification can be done not only over entities, but also over the expressions in the lower-order logics.

Predicate logic provides a formal basis for implementing inductive inference in *Inductive Logic Programming* (ILP). In ILP, descriptions are ordered sets of logical implications restricted to Horn clauses, in which the premise is a product of predicates (conditions), but the consequent (head) can have only one predicate. Practical implementations of ILP employ variants of the PROLOG language. Among the important advantages of ILP are that it has the ability to represent and reason with structured objects, and that it facilitates a reuse of knowledge represented by PROLOG programs (Lavrac and Dzeroski, 1994; Lloyd, 2003). Research in IPL, done primarily in Europe, has shown, however, that ILP learning programs are less efficient than systems employing attribute-based descriptions. Also, knowledge in the form of PROLOG programs can be difficult to understand due to their sometimes complex structure, and due to the use of recursion and unification operators that are not "natural" to most people.

In many applications, the structure of an object is finite and can be described by an attribute-based representation. If the structure is not too complex, its attribute-based representation can be more efficient. To facilitate building descriptions of structured objects, *compound attributes* have been introduced to $\mathcal{A}C$ (Section 6.2).

The basic form of attributional calculus extends propositional calculus by introducing a capability for handling multi-valued attributes, and by directly representing some commonly occurring relations among attributes $\{=, \neq, >, \geq, <, \leq\}$. Also, unlike propositional calculus and first order predicate logic, $\mathcal{A}C$ recognizes and takes into consideration in the inductive generalization process different types of attributes (see rules of generalization described in Michalski, 1983). If all attributes are binary and the extended conditions and additional operators are not allowed, the bare (non-annotated) attributional calculus reduces to propositional calculus.

As its name indicates, attributional calculus is oriented toward creating and reasoning about attribute-based (attributional) descriptions, in contrast to predicate logic, which is oriented toward structural (or relational) descriptions that can involve arbitrary multi-argument relations and functions. $\mathcal{A}C$ can directly express only some, typical relations among attributes. More complex relations can be expressed through the mechanism of derived attributes. Consequently, $\mathcal{A}C$ does not use standard existential and universal quantifiers that apply to variables in the predicate expressions, but rather, it employs a derived attribute, *count* to express universal, existential, and other quantified statements. Due to such forms, and the operators of "exception" and "provided-that", $\mathcal{A}C$ has greater pragmatic representational power for some classes of functions than first order predicate calculus, while, at the same time, it is easier to implement and interpret in natural language. How to efficiently learn expressions with count attributes is, however, an open problem.

As mentioned earlier, $\mathcal{A}C$ statements can be interpreted using a flexible evaluation schema, which evaluates them into continuous degrees of match ("truth"). If the flexible schema evaluates expressions into discrete values (say, k values), $\mathcal{A}C$ can be viewed as a multi-valued (k-valued) logic (e.g., Rine, 1984).

Because of additional logic operators, such as *internal logic operators*, *range* and *exception*, extended conditions and derived attributes, attributional calculus has greater *pragmatic* power and greater *representational* power than propositional logic, multiple-valued logic, and, in special cases, first order predicate logic. The greater "pragmatic power" means that an attributional representation of any function is simpler or equally complex than a logically equivalent representation of this function in another system. The greater "representational power" means that the system can represent some functions that other systems cannot. For example, unlike propositional logic, attributional calculus can represent functions with continuous arguments.

Summarizing, the most important features of $\mathcal{A}C$ from the viewpoint of natural induction include its high expressive power (particularly in relation to decision trees and standard decision rules), the direct correspondence of its expressions to natural language sentences, the availability of both strict and flexible interpretation schemas, and its relatively high efficiency of computer implementation of inductive inference (for hypothesis formation) and deductive inference (for hypothesis testing and application).

6 BASIC CONCEPTS

6.1 Universe of Discourse

As mentioned earlier, $\mathcal{AC}$ is a logic and simple knowledge representation language for implementing inductive, as well as deductive inferences about any entities that can be adequately described in terms of attributes, that is, functions that map entities into single discrete or continuous domains[1]. The entities can be physical or abstract objects, states of objects, situations, processes, behaviors, etc. The set of all entities that are under consideration in a given inference process is called the *universe of discourse.*

The concept of the universe of discourse is very broad. The following examples will illustrate its scope: people, documents, web pages, computer users, social groups, patients, human diseases, drugs, university students, goods in a department store, department stores, customers, behaviors of people or animals, rocks, plants, animals, minerals, songs, stocks, countries, cities, restaurants, military objects and strategies, images, sounds, chemicals, perfumes, etc. These examples suggest application areas in which natural induction programs based on $\mathcal{AC}$ could be useful.

6.2 Attributes and Their Types

A precondition for successful inductive learning about the entities in a universe of discourse is the need to start with an *adequate* set of attributes for describing the entities. A set of attributes is adequate if at least some attributes in it are relevant to the learning problem of interest. An attribute is relevant to the problem if it takes different values for at least some objects of different classes. Based on their discriminatory power for objects of different classes, attributes may have different degrees of relevance. The determination of relevant attributes is normally best done by an expert who has deep knowledge of the domain of application.

In many applications, the information about entities is already stored in some database. This information may include attributes both relevant and irrelevant to the task at hand, and may reside in several relational tables. In such a case, the information in the database is used to create a *target data table*, which contains information viewed as relevant. Because it may be difficult to determine which attributes are relevant to the given problem and which are not, it is better to include in the target data table all attributes that cannot be judged as irrelevant, that is, it is better to commit an error of inclusion and than of omission. This is so because it is easier to detect and ignore or remove irrelevant attributes from a dataset than create missing ones.

For example, in diagnosing diseases, a patient's name is normally assumed irrelevant. However, there can be a situation in which the name is relevant, for example, if the disease occurs mainly in patients of certain ethnic background, which may be indicated by their names. The patient's name is not, of course, a cause of the disease, but may correlate with the disease. The discovery of such correlations can provide useful knowledge for medical treatment and disease prevention.

---

[1] Some authors use the term "features," instead of "attributes," or consider the two synonymous. In our terminology, features are values of attributes. For example, an attribute "color" is not a feature, but its values such as "red" or "blue" are features. Similarly, a continuous attribute "weight" is not a feature of an object, but the statement "the weight is 2 kg" is.

We will assume that in the target data table, attributes correspond to the columns, and individual entities to the rows. The rows contain values of the attributes for each entity. Rows can thus be viewed as descriptions (events or instances) of individual entities. The attributes in the target data table (briefly, data table) span an *instance space (*or *event space*) for entities from the universe of discourse. If the universe of discourse is static, each entity is represented by a single row in the table, or a single point in the instance space. If entities change over time, the universe of discourse is dynamic, and instances move in the instance space.

To represent a dynamic universe, the continuous attribute representing time is discretized into time instances, or time units. A dynamic universe can then be represented by a table in which each entity is mapped into a collection of rows corresponding to its states at different time instances or different time intervals. An alternative representation is to have an ordered collection of data tables, in which each table represents the state of the entities at a given time, or during a given time interval. We thus assume that the starting point of inductive reasoning about entities is a *target* data table describing known entities from the universe of discourse. The set of all entities describable by attributes defined in a data table, e.g., by the fields in the schema of a relational database is called *the universe of discourse spanned by the data table*.

As mentioned earlier, an attribute is a mapping from a set of entities to an *attribute domain*, i.e., the set of all values the attribute can take when applied to entities in the universe of discourse. Such a mapping can be done by a measuring device, human judgment, or some computational process. The structure of the attribute domain determines the *type* of the attribute. The attribute type defines operations that can be meaningfully performed on attributes of that type.

Most of the attribute types recognized in $\mathcal{AC}$ correspond to well-known measurement scales. In addition to the types corresponding to standard measurement scales, such as nominal, ordinal, interval, ratio and absolute, attributional calculus also recognizes cyclic, structured, set-valued, and compound types. Here is a list of the attribute types defined in attributional calculus:

nominal, if the attribute domain is an unordered set (e. g., “blood type,” “person’s name”).

linear, if its domain is a totally ordered set. Linear attributes are in turn classified into *ordinal*, *interval*, *ratio*, and *absolute*, corresponding to the common measurement scales—see Table 1 (for example, a student’s grade is an ordinal attribute, an object’s temperature in ^{o}C is an interval attribute, an object’s weight is a ratio attribute, and the number of atoms in a molecule is an absolute-type attribute).

cyclic, if its domain is a cyclically ordered set (for example, hours of the day, days of the week, months of the year, signs of the zodiac, time zones, etc.).

structured, if its domain is a hierarchically ordered set (for example, a type- or generalization-hierarchy of geometrical shapes, plants, animals, diseases, military ranks, airplanes, etc.).

set-valued, if its domain is the power set of a *base set*, and its values are subsets of the base set. Set-valued attributes can be represented by a set of binary attributes. It is advantageous to use set-valued attributes when the base set is very large, but the subsets constituting their frequent values are small (for example, “patient’s diseases,” where the base set contains possible diseases, or “the set of items purchased by a customer,” where the base set is the set of products in the store).

compound, if its domain is the Cartesian product of the domains of other attributes. A compound attribute is used when an object is characterized by a list of properties of constituent attributes typically associated with this object, but without stating their names. The name of the object is used as the attribute name. For example, "weather" can be used as a compound attribute whose values are lists of properties typically characterizing weather. Compound attributes allow $\mathcal{AC}$ to create expressions that directly correspond to typical natural language descriptions. For example, an expression "weather = sunny & humid" corresponds to a natural language statement "the weather is sunny and humid," which is more natural for people than "the weather-type is sunny and the humidity is yes", a standard logic expression.

The classification of attributes into nominal, ordinal, interval and ratio was introduced by Stevens in 1946. The interval, ratio and absolute attributes are called *quantitative* (also, *metric* or *numeric)*; nominal, ordinal, cyclic, and structured attributes are called *qualitative* (or *non-metric* or *symbolic)*. The "structured" and "cyclic" types were introduced by Michalski (1978b) to facilitate processes of generalizing/specializing descriptions using attributes with domains defined by these types. The compound attribute is a new idea introduced in this paper.

Table 1 provides a summary of attribute types, corresponding attribute domains, and *invariant transformations*. By an invariant transformation is meant a transformation of the attribute domain that does not cause any loss of information conveyed by attribute values.

TYPE/SCALE	***DOMAIN***	***INVARIANT TRANSFORMATION***
Nominal/categorical	**unordered set**	**isomorphic**
Structured	**partially ordered set**	**node-type dependent**
Set-valued	**partially ordered set**	**isomorphic**
Ordinal	**totally ordered set**	**monotonic**
Cyclic	**cyclically ordered set**	**monotonic-cyclical**
Interval	**totally ordered set**	**linear: y'= a + by**
Ratio	**totally ordered set**	**ratio: y'= ay**
Absolute	**totally ordered set**	**none**
Compound	***depends on the type of constituent attributes***	

Table 1: Types of attributes recognized in Attributional Calculus.

Simple examples will help to explain the above attribute types. Suppose that the domain of a nominal attribute "color" is {red, blue, yellow}. By isomorphically mapping this domain to any other three-element set, e.g., {czerwony, niebieski, zolty}, we can use attribute color with new values without any loss of information, as long as the mapping is applied consistently. Meaningful statements involving such variables use "equal" and "not-equal", e.g., color = red, or color $\neq$ blue. All the standard logical operations apply to such statements, such as negation, logical disjunction, logical conjunction, exclusive-or, and equivalence. If one maps the domain to a set of numbers, e.g., {1, 2, 3}, these numbers serve only as labels, and cannot be meaningfully used in arithmetical operations.

In structured attributes, the partial order is defined by a generalization hierarchy (a.k.a. *type* or *is-a* hierarchy) in which every parent node represents a more general concept than its offspring nodes. Examples of such hierarchies include shapes, geographical names, plants, animals, etc. Figure 1 presents an example of a possible domain of the structured attribute "shape."

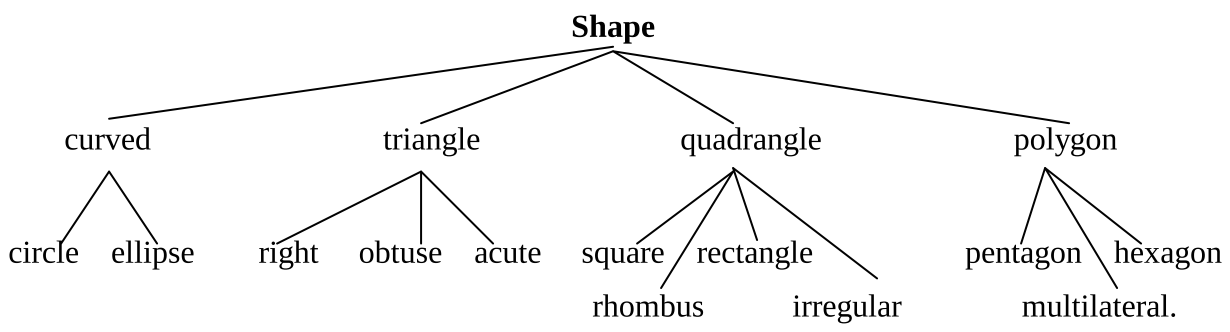


Figure 1: The structured attribute "shape" and its domain.

In addition to operators applicable to nominal attributes (attribute = value, attribute ≠ value), operators applicable to structured attributes include *structure-based generalization*, *specialization*, and *modification*. The generalization operator, *climb-tree(A, V, k)*, replaces value *V* in a description involving attribute *A* by a more general value located *k* levels up in the generalization hierarchy for this attribute. For example, *climb-tree(shape, obtuse, 1*) replaces value *obtuse* by *triangle* in a statement involving the attribute *shape.*

The operator *descend(A, V, k)* is a non-deterministic specialization operator that replaces value *V* by one of the values located *k* levels down from *V*. The value is selected among alternatives either randomly or algorithmically. For example, the operator *descend(shape, triangle, 1)* may randomly generate the value *right triangle* as an instance of a triangle. Finally, the operator *select-sibling(A, V)* is a generate-alternative operator that replaces *V* by one of its sibling values in the hierarchy (again, the value can be selected randomly or algorithmically). For example, *select-sibling(shape, square)* may produce "rhombus." As the above examples show, structured attributes facilitate operations of description generalization, specialization, and modification.

The domain of ordinal (or rank) attributes is a totally ordered discrete set. For example, a course grade, (A, B, C, D or F) is a rank attribute. A discretized continuous attribute is also an ordinal attribute. For example, the range of a continuous attribute "blood pressure" can be discretized into {1, 2, 3}, where "1" stands for "low", "2" stands for "normal" and "3" stands for "high". If we perform a monotonic transformation of this domain into <3, 59, 70>, where "3" corresponds to "low", "59" to "normal", and "70" to 'high", the numerical order of the values remains the same; we can therefore use the new domain without any loss of information, since values of ordinal attributes only convey information about the order, and not about the magnitude.

Operators applicable to ordinal attributes include not only those applicable to nominal and structured attributes, but also relational operators, <, ≤, >, ≥, and the *range* operator, denoted by "..", which defines a range of values. For example, if the domain of an ordinal attribute "strength" is {very_weak, weak, medium, strong, very_strong, mighty_strong}, we can write, *strength* > *medium*, to say that the value of *strength* is greater than *medium*, or *strength* = *weak .. very_strong*, to say that the value is between *weak* and *very strong*, inclusive.

Set-valued attributes are useful for describing situations in which values of interest are small subsets of a large domain, called the *base set*. For example, the attribute *itemset* is a set-valued attribute whose values are sets of goods purchased by a customer in a department store. Individual customers usually purchase only a very small subset of goods available in a department store (the latter is the base set). The domain of set-valued attributes is a lattice.

A set-valued attribute can be formally replaced by a set of binary attributes, each corresponding to a single element in the base set. Such a representation may not be practical, however, because it would have to include as many binary variables as there are elements in the base set (which can be a very large set), while the number of binary variables with value 1 would typically be small. Values of set-valued attributes can be used as operands in set-theoretic operations, such as union, product, and set-difference, and in set-theoretic relations $A \subseteq B$, $A \subset B$, $A \not\subset B$, $A \cap B = \emptyset$. For example, given statements *itemset=A1* and *itemset=A2*, one can make statements such as $itemset=A1 \cup A2$, $itemset=A1 \cap A2$, or *itemset= A1-A2.*

Cyclic attributes are like rank variables, but assume that the last value is followed by the first value in their domain. The operators >, ≥, < and ≤ do not apply to these attributes, but the range operator ".." does. Cyclic attributes are used to describe cyclically occurring phenomena, e.g., days of the week or months of the year. Thus, one can write: *day_of_week* = *Wed..Mon*.

Interval attributes characterize quantities on a scale in which "0" is arbitrary, and express counts of some arbitrary units. For example, the temperature can measured on the Celsius scale (^{o}C), where "0" denotes the temperature of freezing water, and "100" denotes the temperature of boiling water, or the Fahrenheit scale (^{o}F), which is related to the Celsius scale by a linear function: $t^oC = 5/9\,(t^oF - 32)$, where t^oC and t^oF are temperatures measured on the Celsius scale and the Fahrenheit scale, respectively. Any other linear transformation of the domain will not change the meaning of the temperature measurement, as long as it is applied consistently.

In addition to the previous operations, interval attributes are also amenable to operators "+" and "-". For example, one can say that the difference between the temperature today, t_1, and yesterday, t_2, is $\Delta t\,^oC = t_1\,^oC - t_2\,^oC$, which can be equivalently expressed as $F(\Delta t\,^oC) = F(t_1\,^oC) - F(t_2\,^oC)$, where $F(t^oC)$ denotes temperature in ^{o}F equivalent to temperature in ^{o}C. The multiplication and division are not invariant operators for interval attributes. For example, $F(3 * t^oC) \neq 3 * F(t^oC)$). Thus temperatures measured in ^{o}C or ^{o}F can be meaningfully added or subtracted, but not divided or multiplied. For example, if the temperature in a room rose from 35^{o}F to 70^{o}F, it does not mean the room is twice as hot.

Values of ratio attributes represent counts of some predefined units, as in interval attributes, but zero is not arbitrary; rather, it denotes the absolute minimum value. Units can be, for example, kilograms or pounds for a weight attribute, inches or centimeters for a length attribute, etc. Different units are related by a multiplication coefficient. In addition to all the previous operators, ratio attributes can be also be used in all arithmetic and algebraic operations. Absolute variables represent counts of some items in a set (e.g., the number of people in a building). There have no units; and no transformation is allowed. Absolute attributes are amenable to all operations to which ratio attributes are.

Compound attributes have been introduced to concisely describe objects or situations characterized by constituent attributes, which are applicable only to these objects. Values of compound attributes are internal conjunctions of values of constituent attributes. For example, attributes applicable to describing a house can be the number of bedrooms, the size of the kitchen, of the backyard, etc. To describe individual bedrooms, other attributes are needed, e.g., the length and width of the bedroom, the number of windows, and the closet size. In attributional calculus, the attributes related only to the bedroom would be combined into one compound attribute called "bedroom." Using the attribute bedroom, one can make a statement: bedroom = 20' long & 13' wide & 3 windows & large_ closet.

In addition to regular values, every attribute domain is also assumed to contain three *meta-values*. These values represent possible responses to questions requesting the attribute value for a given entity, specifically: "don't know" (denoted by "?"), "not applicable" (denoted by "NA"), and "irrelevant" (denoted by "*"). The meta-value "don't know" is given to an attribute whose value for a given entity is unknown for whatever reason; for example, it has not been measured or has not been recorded in the database. "NA" is given to an attribute that is not applicable to a given entity. For example, the attribute "the number of pages" is not applicable to a chair in the library, but is applicable to a book. The attribute "the person's shoe size" can be viewed irrelevant to the problem of determining the person's education level. This is indicated by assigning to it the meta-value "*".

Because the attribute type is associated with operators applicable to them, the attribute type is useful in conducting inductive inference. By applying generalization rules according to the attribute type, an inductive generalization process can produce more plausible generalizations, and, by avoiding implausible generalizations, can also be more efficient. The types and domains of attributes are parts of the background knowledge about the given inductive generalization problem. A collection of inductive generalization rules applicable to statements with attributes of different types was presented in (Michalski, 1983).

6.3 Event Space and Concept Space

Let $\mathcal{U}$ denote the universe of discourse, and $\mathcal{X}$ the set of attributes applicable to entities in $\mathcal{U}$ and considered relevant to the problem of interest. A vector of values of attributes from $\mathcal{X}$ (or, equivalently, a vector of attribute-value pairs) describing some entity (an object, a situation, a process, etc.) is called an *event*. The entity described by an event is called *an instance*. When it leads to no confusion, the terms "event "and "instance" are used interchangeably. The set of all possible events, $\mathcal{E}(\mathcal{U}, \mathcal{X})$, or briefly, $\mathcal{E}$, is called the *event space* (*instance space* or *representation space*) *spanned over* $\mathcal{X}$ for the universe $\mathcal{U}$.

The event space is thus determined by the Cartesian product:

$$\mathcal{E} = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots . \times \mathcal{D}_n \qquad (1)$$

where $\mathcal{D}_1$, $\mathcal{D}_2$,…, $\mathcal{D}_n$ are domains of attributes in $\mathcal{X}$.

An arbitrary event--a point in the representation space--may represent just one entity, a group of entities, or no entity from the universe of discourse $\mathcal{U}$. If all attributes in $\mathcal{X}$ are discrete, $\mathcal{E}$ is finite; otherwise, it is infinite.

A concept can be viewed as any set of entities that is treated as a single unit in reasoning. The entities belonging to the concept are called concept instances. Among the possible reasons for combining entities into concepts are their similar appearance (e.g., the concept of a “circle”), their common function (e.g., the concept of “chair” or “food”), or their similar role or behavior in a larger system (e.g., the concept of a “planet”, an “electron”, or an “employee”). Concepts are typically given names, so that they can be referred to easily. Given an event space for a universe of entities, concepts are represented as subsets of that space.

In concept learning, only a limited set of instances of a concept is typically known. Concept instances that are available for learning are called training examples, and represented as pairs <instance, concept name>, where the concept name is a name given to the concept, or a symbol identifying the concept. Concept names are assigned by a teacher, or, generally, by an “oracle” that already knows the concept.

Concepts to be learned are called *target concepts*, and subspaces corresponding to them in the instance space are called target concept representations, or, simply, target concepts, when this causes no confusion. Target concepts may potentially contain an infinite number of instances, and their boundaries in the representation space may be subjective and context-dependent. Think, for example, of all possible instances of the concept “face” or “animal.”

In learning a single concept, the training examples that represent instances belonging to the concept are called positive examples (concept examples), and those not belonging to the concept are called negative examples (counterexamples). The ultimate goal of learning is to create an accurate target concept description. Because the training set is typically only a subset of the target concept, in practice, all that a learning system can do is to seek a good approximation of it.

A learned description is called a *complete training set [target] description* if it describes all positive training examples [all target concept instances], and possibly also some other examples. A description is called a *consistent training set [target] description* if it describes no negative examples [no non-target instances]. A description is called consistent and complete (CC) training set [target] description if it describes all positive training examples [all target concept instances], and no negative examples [no non-target instances]. The latter definition assumes that the training set does not have inconsistent (ambiguous) examples, that is, examples that can be both positive and negative.

A description is called an *approximate training set [target] description* if it partially violates one or both of the completeness and consistency criteria. The set of all CC training set descriptions is called the version space; the set of all approximate training set descriptions is called an approximate version space. Because concept instances are perceived through the values of the attributes characterizing them, a concept defines a subarea of the event space spanned over these attributes. Concept learning can be thus characterized as determining a description defining this subarea. If an event falls into this subarea, it is recognized as an instance of the concept.

Let $\mathcal{D}$ denote a set of concepts to be learned. Typically, $\mathcal{D}$ is a set of concepts without any particular order, for example, a set of alternative decisions to be taken in a given situation. In some applications, $\mathcal{D}$ can be a totally ordered set (e.g., ranks of universities), or a partially ordered set (e.g., a hierarchically ordered set of diseases). Therefore, it is convenient to view **D** as the domain of an output attribute, y, which is assigned a type depending on the learning problem. A so-defined output attribute can be an input attribute for another concept learning problem. The training set for $\mathcal{D}$ is a set of pairs {event, concept}, which link an instance with the concept it belongs to. This training set can be viewed as a function:

$$f_t\colon \mathcal{E} \rightarrow \mathcal{D} \cap \{*\} \tag{2}$$

where "*" is assigned to events not specified in the training set ("don't know"). Learning concepts from $\mathcal{D}$ means developing a description in a given language characterizing a mapping:

$$f\colon \mathcal{E} \rightarrow \mathcal{D} \tag{3}$$

The mapping (3) asserts that every event in $\mathcal{E}$ belongs to some concept in $\mathcal{D}$. To assure that this assumption is always correct, $\mathcal{D}$ should include a value representing "none of the other concepts."

A training set typically assigns values from $\mathcal{D}$ to only a small subset of $\mathcal{E}$; hence there are many instances to which f_t assigns the value "*". The space of all functions specified by (2), i.e., the set of all completely or incompletely defined functions mapping events into concepts is called the *concept mapping space* spanned over the given event space, denoted CM($\mathcal{E}$). For illustration, Figure 2 visualizes an event space spanned over binary variables, x_1 and x_2, and the training data for a very simple concept. Each cell represents one combination of attribute values. Cells marked by 0, 1, or * represent negative, positive or undetermined concept examples, respectively.

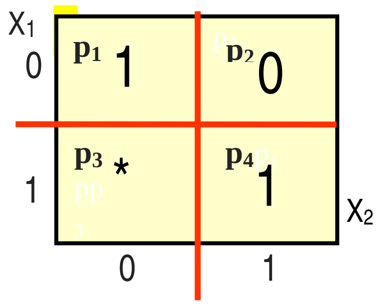
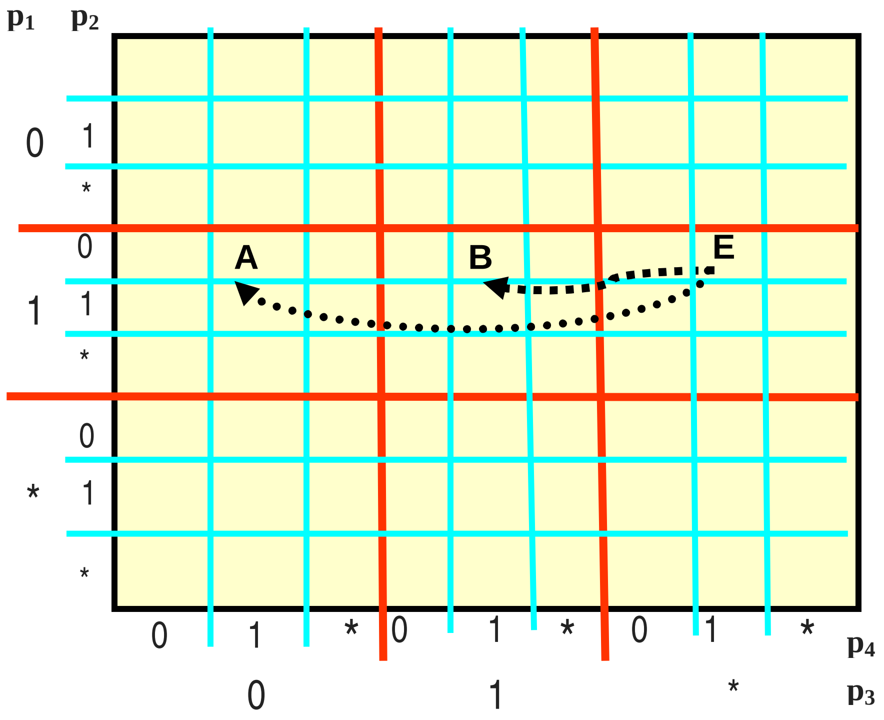


Figure 2: A visualization of a very simple event space and training events.

Figure 3 visualizes the concept space spanned over this representation space (i.e., the set of all completely or incompletely defined functions of the attributes). The visualization method used here employs the *Generalized Logic Diagram* (GLD; Michalski, 1978a, Wnek, 1995). The event space in Figure 2 is spanned over two binary attributes x and y, whose values are assigned to rows and columns, respectively. Each cell represents one combination of attribute values (one instance or event). Cells marked by a "1" represent positive examples of a concept, and the cell marked by a "0" represents a negative example.

Each assignment of values 0, 1, or * to cells in the diagram in Figure 3 represents one instantiation of function (2). There are 3^{2^n} different instantiations, where n is the number of attributes. In this case the concept space thus has 81 points. The cell marked by E represents the training data defined in Figure 2 (vector 1, 0, *, 1), and cells A and B represent two different generalizations of the training data. In Figure 3, attributes **p_1, p_2. p_3, and p_4** represent cells in Figure 2, and the values 0, 1, and * represent values assigned to the cells by a function.



"A" and "B" represent assignments of 0 and 1, respectively, to * in the representation space.
Figure 3: A visualization of the concept space spanned over the instance space in Figure 2.

6.4 Description space

The objective of concept learning is to create descriptions delineating subspaces in the event space that represent concepts to be learned. To learn a concept description, one needs to use a description language. A great variety of concept description languages have been used in research in machine learning and data mining. Among the most common are decision trees, decision rules, predicate logic, support vector machines, Bayesian nets, semantic nets, and frames. Given a description language and a representation space, there can usually be many different descriptions of the same concept.

Let $\mathcal{L}$ be a description language and CM($\mathcal{E}$) the concept mapping space that maps $\mathcal{E}$ into $\mathcal{D} \cap \{*\}$. The set of possible descriptions in $\mathcal{L}$ of functions in the concept mapping space is called the *description space* for $\mathcal{D}$ in $\mathcal{L}$, and denoted DS($\mathcal{D}$, $\mathcal{L}$). For example, if $\mathcal{L}$ is the language of decision trees, then the set of all possible decision trees describing all functions in the concept mapping space for $\mathcal{D}$ is the description space DS($\mathcal{D}$, $\mathcal{Decision\_Trees}$).

Concept learning can thus be viewed as a process of traversing a description space, starting with a description of the training data and ending with a description of the target concept to be learned. The fundamental difficulty of concept learning is that determining a description of a target concept from a partial concept description given by the training data is an under-constrained problem. Therefore, there can be many ways to generalize the training data, and each such generalization can be represented in many ways in a given description language.

A typical solution to this problem is to determine a description of a complete function in the concept mapping space that is consistent (or nearly consistent) with the training data and satisfies some extra-logical criterion. In symbolic learning, such a criterion may be to determine the simplest description in the given language. In support vector machines, it may be a maximal margin hyperplane (Scholkopf and Smola, 2002).

6.5 An Illustration of Hypothetical Concept Descriptions

To illustrate the ideas presented above, let training events be vectors of four binary attributes x, y, z and w, and the description language be propositional rules. The training set for learning is presented in the event space illustrated by the diagram in Figure 4a. Pluses and minuses denote positive and negative concept examples, respectively.

Different hypothetical concept descriptions learned from the training data are visualized in Figures 4b, 4c and 4d. Each shaded subarea in a diagram represents one propositional decision rule. A set of the rules shown in a diagram represents a concept description. Concept descriptions presented in Figures 4b, 4c and 4d represent hypothetical generalizations of the training data in Figure 4a.

Using the diagrams, one can write down propositional rules expressing S1, S2, and S3:

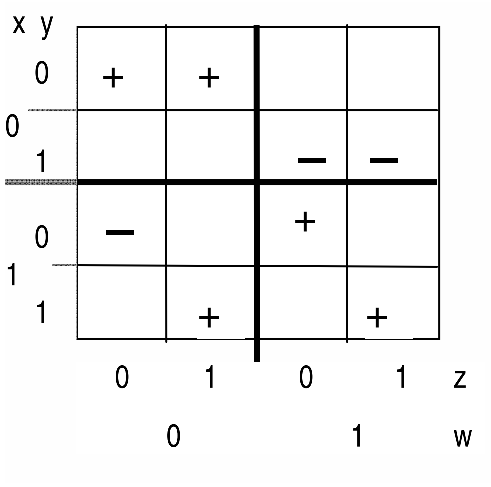
S1: ***C*** <= ~x~y, ***C*** <= xz, ***C*** <= x~yw

S2: ***C*** <= ~x~z~w, ***C*** <= xyzw, ***C*** <= z~w, ***C*** <= x~zw, ***C*** <= x~yw

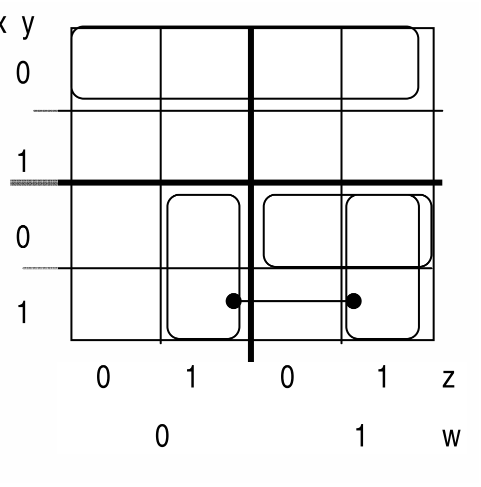
S3: ***C*** <= ~x~w, ***C*** <= z~w, ***C*** <= xw

Descriptions S1 and S2 represent significantly different concepts. Descriptions S2 and S3 represent exactly the same concepts, but differ significantly in complexity (measured by the number of rules and literals). It is not clear how to choose between S1 and S3. S3 is a little simpler than S1 in terms of the number of literals (6 versus 8), but this difference does not mean that S3 will be more correct than S1 when applied to new data. Using rules of logic, one can show the logical equivalence between S2 and S3, but such a process can, in general, be very complex and time-consuming, especially when there are many attributes. Finding the equivalence between two expressions or determining the minimum expression of a logic function is NP-hard.

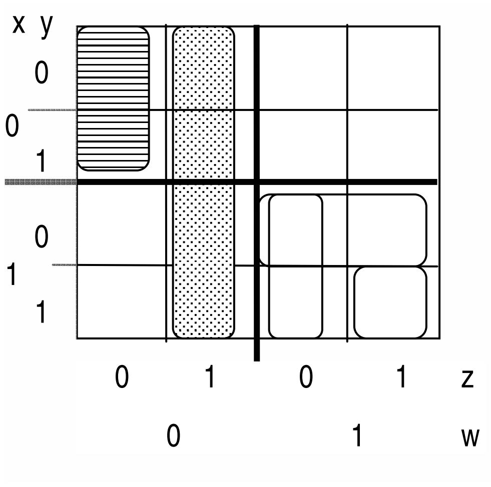
The above example shows that even a very simple training set can yield significantly different generalizations, and that descriptions of exactly the same concept in the same representation language can vary significantly in complexity.



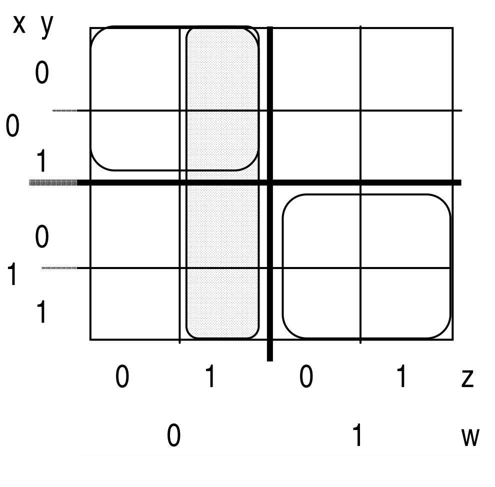
a) Visualization of training data



b) Concept description S1



(c) Concept description S2



(d) Concept description S3

Figure 4: Training data and different generalized descriptions.

6.6 Learning Error

Given an event space $\mathcal{E}$ spanned over attributes $\mathcal{X}$, let $\mathcal{TC}(e)$, $e \in \mathcal{E}$, denote a target concept description (i.e., the correct concept description). $\mathcal{TC}(e)$ represents a mapping of $\mathcal{E}$ into {0,1}, where "1" means that ***e*** is a member of the target concept and "0" that it is not. Let $\mathcal{LC}(e)$ denote the learned concept description, which is a mapping of $\mathcal{E}$ into {0,1}, where "1" means that *e* satisfies the learned description, and "0" that ***e*** does not satisfy it. Let P($\mathcal{E}$) be a probability distribution of events in $\mathcal{E}$. It tells us how frequently we see different events *e* from $\mathcal{E}$.

When an event $e \in \mathcal{E}$ is seen, the probability that $\mathcal{LC}(e)$ will assign to it an incorrect concept is:

$$\text{pError}(e) = \text{p}(\mathcal{LC}(e) \neq \mathcal{TC}(e)) \tag{4}$$

The *learning error* of the learned concept description $\mathcal{LC}(e)$, denoted lError, is thus:

$$\text{lError} = \sum_{P(\mathcal{E})} \text{pError}(e) \,/\, |\mathcal{E}| \;=\; \sum_{P(\mathcal{E})} p(\mathcal{LC}(e) \neq \mathcal{TC}(e)) \,/\, |\mathcal{E}| \qquad (5)$$

where $|\mathcal{E}|$ denotes the size of the event space.

6.7 Relevance and Adequacy Criteria for the Attribute Set

It is well-known that the success of concept learning is crucially dependent on the attributes used in the formulation of the problem (e.g., Bongard, 1967; Liu and Motoda, 1998). If these attributes are irrelevant for the learning task, no method will be able to learn a correct concept description. For example, if the task is to learn a distinction between excellent students and other students, and the attributes are the length of the student's nose and the color of the student's notebook, then it will be impossible to learn a reliable description, except perhaps for a very unusual situation.

In order to get a better insight into this issue, let us classify attribute sets into different types. If every entity from the universe of discourse, $\mathcal{U}$, maps into a unique point in the instance space, $\mathcal{E}$, spanned over an attribute set $\mathcal{X}$, then $\mathcal{X}$ is called *isomorphic*; otherwise, it is *homomorphic* for $\mathcal{U}$. For example, if $\mathcal{U}$ is a set of patients, and each patient can be uniquely described in terms of attributes in $\mathcal{X}$, then $\mathcal{X}$ is isomorphic for that universe.

If an attribute set, $\mathcal{X}$, maps different (non-intersecting) concepts into disjoint subsets of the instance space, then the instance space spanned over $\mathcal{X}$ is called *consistent*, otherwise, it is called *inconsistent*. If the instance space is consistent, it is always possible to create a consistent and complete (CC) concept description with regard to a training set (given a sufficiently expressive description language). Such a description classifies instances of the training set perfectly, but may not correctly classify unseen instances. If the instance space is inconsistent, it is impossible to learn a CC concept description. In such cases, only *approximate descriptions* can be learned.

An isomorphic attribute set always spans a consistent instance space. A homomorphic attribute set may span either a consistent or an inconsistent instance space. If it always maps entities of different concepts into different points in the space, then the space is consistent (Figure 5). If it maps some entities of different concepts into the same points in the space, then the space is inconsistent (Figure 6).

Attribute sets can be classified on the basis of how relevant they are for the learning task (the concepts to be learned), and how adequate they are for the learning method used, or, more precisely, for the description language employed in the method. The first question concerns the *relevance criterion*, and the second concerns the *adequacy criterion* for the attribute set. If we make the simplifying assumption that both criteria are strict, that is, an attribute set either satisfies or does not satisfy them, there are potentially four combinations of answers.

Only three are meaningful: 1) the attribute set is relevant for the learning task and adequate for the method employed, 2) the attribute set is relevant but inadequate, and 3) the attribute set is irrelevant and inadequate. There cannot be an attribute set that is irrelevant but adequate.

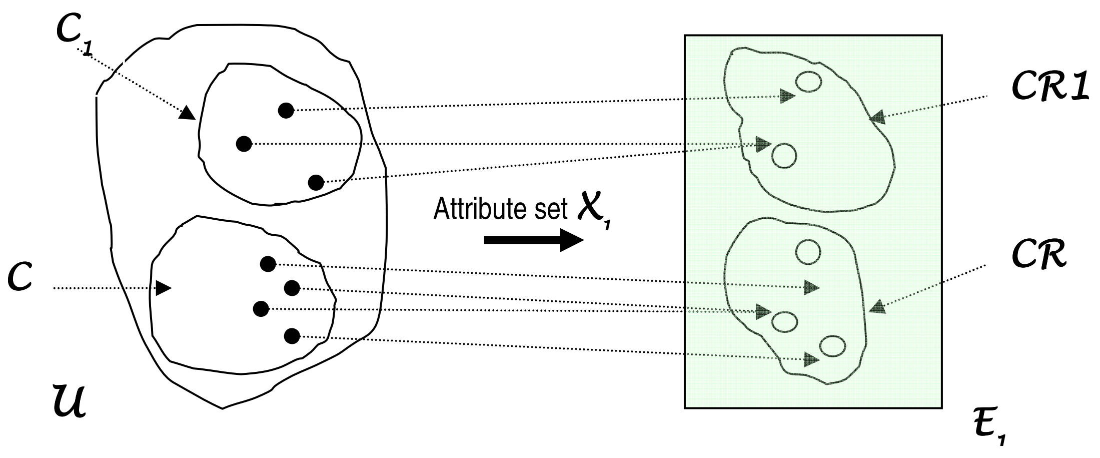


Figure 5: A homomorphic attribute set $\mathcal{X}_1$ spanning a consistent instance space.

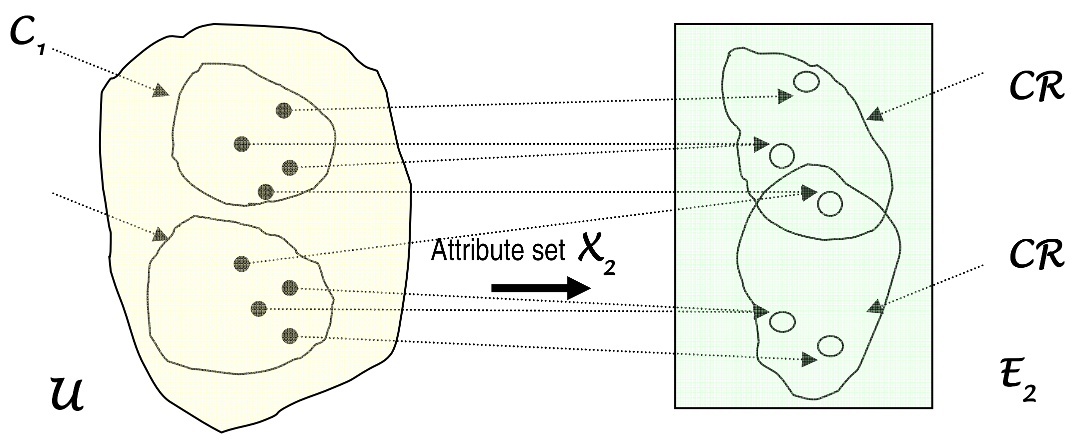


Figure 6: A homomorphic attribute set $\mathcal{X}_2$ spanning an inconsistent instance space.

To explain these criteria, let $\mathcal{D}$ denote a set of target concepts (the domain of an *output attribute*), and M denote a learning method to learn these concepts. The method M may employ, for example, simple rule-based or decision tree description languages (which draw axis-parallel hyper-rectangles in the instance space), support vector machines (which draw hyper-planes in a transformed instance space), or some other representation system.

In concept learning, there is a distinction between target concepts, $\mathcal{D}$, and training sets for them, $\mathcal{D}_{tr}$. The goal of learning is to induce a description of target concepts from their training sets. Let us assume that $\mathcal{D}$ consists of disjoint concepts, that is, concepts that do not have common instances, and $\mathcal{X}$ consists of *intrinsic* attributes, that is, attributes capturing measurable aspects of concept instances, rather than arbitrary labels or names assigned to instances.

An attribute set $\mathcal{X}$ is called *intrinsically relevant for* $\mathcal{D}$ (or $\mathcal{D}$*-relevant*) if it consists of intrinsic attributes, and representations of concepts from $\mathcal{D}$ do not intersect in the $\mathcal{X}$-spanned instance space. An attribute set $\mathcal{X}$ is called *perceptibly relevant* (or $\mathcal{D}_{tr}$*-relevant*) if representations of training sets of concepts from $\mathcal{D}$ do not intersect in the $\mathcal{X}$-spanned space. If in the above definitions the concept representations *partially intersect*, the term "*relevant*" is replaced by "*partially relevant*".

An intrinsically relevant attribute set may include some irrelevant attributes, but it must include also some relevant ones for learning concepts in $\mathcal{D}$. An attribute set is called *irreducibly intrinsically relevant*, if none of its subsets is intrinsically relevant.

Note that an attribute set can differentiate between instances of different concepts, but may be useless for creating concept descriptions if it contains non-intrinsic attributes. For example, the attribute "patient's name" in a dataset of liver cancer patients identifies each patient, and thus discriminates between liver cancer and non liver cancer patients, but is useless for diagnosing the liver cancer. This is so because it is a non-intrinsic attribute, as its values are not measurable properties of patients.

Given a learning method M and a set of target concepts $\mathcal{D}$, an attribute set is called *adequate* for M if it is intrinsically relevant for $\mathcal{D}$, and the description language used by M is "adequate" for describing boundaries between concepts in the representation space. The word "adequate" is not precise, but it means here that the description of the concept will not be "overly" complex (according to some reasonable measure of description complexity). To explain this subjectively defined concept, suppose the concept boundary is a circle, and the method can only draw axis-parallel rectangles in the instance space. To describe the concept, the method will have to use a large number of tiny rectangles. We call such a method not adequate for this learning problem.

Note also that an attribute set may be relevant for the training set, but irrelevant for the target concept. Given a learning problem, defined by the training examples expressed in terms of the original attribute set $\mathcal{X}$, one usually does not know if the given attribute set is intrinsically relevant; one may only determine if it is perceptibly (fully or partially) relevant. When defining an original attribute set for concept learning, it is thus important to define at least a perceptibly relevant attribute set. If this is impossible, then concept learning has to employ statistical or other methods for learning in inconsistent event spaces (e.g., Duda and Hart, 1973; Michalski and McCormick, 1972; Pawlak, 1991).

To illustrate the ideas introduced above, consider target concepts "Odd-sum" and "Even-sum", defined as "sets of integers whose binary representation contains an odd number of "1s" and an even number of "1s", respectively. Assume that training sets of both concepts consist of decimal numbers, and that all "Odd-sum" integers happen to be written in red, and all "Even-sum" integers happen to be written in blue. Assume also that most "Odd-sum" integers in the training set are typed using the Arial Narrow font, and the most of the "Even-sum" integers are typed using the Lucida Handwriting font.

The attribute set $\mathcal{X}_1 = \{x_1, x_2, x_3, \ldots, x_n\}$, where x_i are digits in the binary representation of the integers, is both intrinsically and perceptibly relevant to the learning problem, because concept representations and training set representations of the concepts do not intersect in the space spanned over $\mathcal{X}_1$ (unless there are errors in the data). The set $\mathcal{X}_2$ = {font_color} is perceptibly relevant, but is not intrinsically relevant, because in some future concept instances, colors may not be correlated with the concepts. The set $\mathcal{X}_3$ = {font_type} is partially perceptibly relevant but not intrinsically relevant. The union of the three attribute sets, $\mathcal{X} = \{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3\}$ is both intrinsically and perceptibly relevant.

Attributes such as "the parity of integers in the decimal system," "the number of decimal digits," "the size of the digits," "the type of ink used to type numbers down," etc. are not intrinsically relevant, and likely not perceptibly relevant (if their values do not have a strong accidental correlation with the concepts). Suppose that the chosen learning method M expresses concept descriptions in the form of *atomic rules*, defined as logical conjunctions of conditions: *attribute = value*. For the problem presented above, a learning system may create a set of rules:

{If font_color = red, concept = Odd-sum; If font_color = blue, concept = Even-sum} (6)

These rules in (6) would perfectly differentiate between instances of two concepts in the training set, and might also perform relatively well in classifying any future concept instances if the font color remains sufficiently correlated with the concepts. In this case, the attribute "font_color" is *adequate* for the given method.

Suppose now that the training set did not include the attribute font_color, but rather the attribute set $\mathcal{X}_1 = \{x_1, x_2, x_3, \ldots, x_n\}$, as defined above. Given that the training set is finite, the method would still find a set of atomic rules that differentiate training examples of both concepts, but the description would be quite complex in terms of the number of rules, and would likely not work well on future examples. Therefore, the attribute set $\mathcal{X}_1$ is inadequate for the method, though it is relevant for the target concepts.

If the learning method M also has the ability to sum-up the binary digits, and to determine the parity of the sum, the concept description could then be quite simple. For example, if the method could construct a new attribute $\mathcal{X}' = \text{Parity}(\text{Sum}(x_1, x_2, x_3, \ldots, x_n)$, whose value is 1 if the parity of the sum of binary digits x_i is odd, and 0 if it is even, it could then generate the concept description:

If $\mathcal{X}' = 1$, then concept = Odd-sum; If $\mathcal{X}' = 0$, then concept = Even-sum (7)

The attribute set $\{\mathcal{X}'\}$ is thus relevant for the target concepts, and adequate for the method.

Suppose now that M uses a representation language of decision trees. In this case, the attribute set $\mathcal{X}_1$ is neither $\mathcal{D}$-adequat*e* nor $\mathcal{D}_{tr}$–adequate; $\mathcal{X}_2$ is not $\mathcal{D}$-adequate, but it is $\mathcal{D}_{tr}$-adequate, $\mathcal{X}_3$ is not $\mathcal{D}$-adequate, but it is approximately $\mathcal{D}_{tr}$-adequate, and the attribute set $\{x_1, x_2, x_3, \ldots, x_n, \mathcal{X}'\}$ is relevant for target concepts and for training sets, and adequate for the method.

As illustrated in the above example, an attribute set may be relevant for the target concept, but not adequate for the method. In such cases, areas in the original instance space representing individual concepts have boundaries that are difficult to describe using the given representation language. For example, if the target set forms a rhombus in the instance space, then the attribute set spanning the instance space is inadequate for methods that "speak" in terms of axis-parallel rectangles. In this case, one needs either to apply a method for which the attribute set is adequate, or span the instance space over different attributes.

An attribute set can become adequate if the method can generate additional, *derived attributes*, which are functions (arithmetic, logical, or other) of the original ones. The problem of generating adequate derived attributes is a topic in *constructive induction* (e.g., Bloedorn and Michalski, 1998). Specifically, constructive induction methods aim at discovering derived attributes that span an instance space in which concept boundaries are much more regular than in the original space. Figure 7 illustrates the relationship among the Universe of Discourse, Original Representation Space and Derived Representation Space[2].

Derived attributes can be characterized by their *order*. The first order derived attributes are functions of the original ones (zeroeth order). Such functions may be, for example, linear functions of numerical attributes (as in principal component analysis), any arithmetic functions of numerical attributes, logical expressions involving nominal attributes, Fourier transforms, wavelet functions, etc. The kth order attributes are functions of sets of attributes among which there is at least one of (k-1)th order.

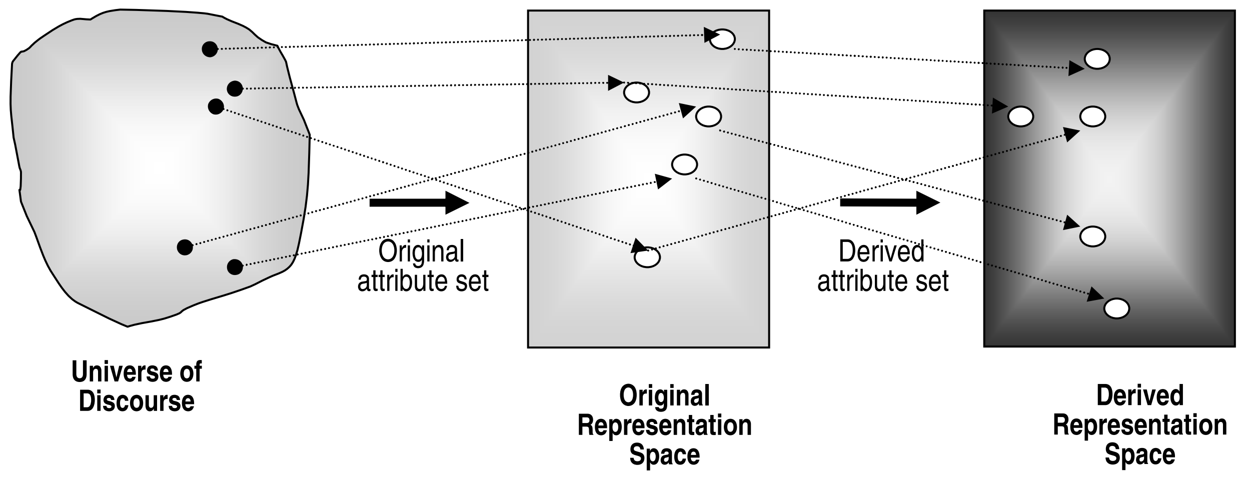


Figure 7: The universe of discourse and original and derived instance spaces.

7 DEFINITION OF ATTRIBUTIONAL CALCULUS

Attributional calculus is a modification and extension of the *variable-valued logic one* (VL1), a form of multiple-valued logic (Michalski, 1972; 1974). It is defined by its syntax, semantics, cognitive constraints, and an evaluation schema. It is not just one fixed system, but has several forms. The more advanced the form, the richer the representation system, but the more difficult it is for implementing inductive inference.

The concept of "cognitive constraints" is novel in the definition of a logic system, and is introduced in order to reflect the objective of attributional calculus to serve as a representation system for natural induction. An attributional calculus system is a 4-tuple:

<syntax, semantics, cognitive constraints, evaluation schema> (8)

[2] Some authors call the original event space an *input space*, and the derived representation space a *feature space*.

The syntax specifies formally correct $\mathcal{AC}$ expressions. The $\mathcal{AC}$ semantics depends on the interpretation schema. $\mathcal{AC}$ can have many interpretation schemas.

The following sections describe individual components of the definition of attributional calculus. We start with the basic syntactic and semantic construct called an *attributional condition*.

7. 1 Attributional conditions

The basic representational unit of attributional calculus is *attributional condition* (or *selector),* which roughly corresponds to a simple sentence in natural language that describes an entity or a group of entities in terms of one, or at most few attributes. A selector relates one or more attributes to their values or other attributes. A general form of a selector is:

[L rel R],

where:

L (the left side or referent) contains an attribute, or a group of attributes with the same domain. Attributes from the group are joined by "&" or "v", called internal conjunction and disjunction, respectively. ***L*** can also utilize one of the standard derived attributes: *count*, *max*, *min,* and *avg*.

R (the right side or reference) is an expression specifying a value or a subset of values from the domain of the attribute(s) in ***L***. If the subset contains values of a nominal attribute, they are joined by the symbol "v" (called *internal disjunction*); if the subset contains consecutive values of a linear attribute, they are represented by joining the extreme values by operator "..", called *range*. ***R*** can also be an attribute with the same domain as the attribute or attributes in ***L***.

rel is a relational symbol from the set: $\{=, \neq, >, \geq, <, \leq\}$. Relational operators $\{=, \neq\}$ apply to all types of attributes. Relations $\{>, \geq, <, \leq\}$ apply only to linear attributes.

Brackets **[ ]** can be written as (), or omitted if their omission causes no confusion. If brackets are used, the conjunction of two selectors is usually written as their concatenation. If an attribute, x, is binary, the condition [x = 1] can be written simply as the literal x, and [x= 0] as the literal ~x. Thus, if attributes are binary, attributional conditions reduce to propositional literals. An attributional condition is called *basic* if its left side, ***L***, is a single attribute, the relational symbol is one of $\{ = \; > \; \geq \; < \; \leq \}$, and the right side, ***R***, is a single value; otherwise, it is called *extended*.

Attributional conditions are interpreted in the context of the entity (object, situation, etc.) to which they are applied. They can have a strict (propositional) interpretation, or flexible (or multi-valued) interpretation. In the strict interpretation, ***[L rel R]*** is said to be satisfied by the entity (or evaluates to true) if **L** is in relation **rel** to **R** for that entity.

In the multi-valued interpretation of an attributional condition, ***[L rel R]*** is said to be satisfied or matched by the entity to degree α (or, simply, to evaluate to a degree α) if for that object ***L*** is in relation ***rel*** to ***R*** with the degree α, according to a given flexible evaluation schema (Section 9).

Examples of attributional conditions with their strict natural language interpretation are presented below. An underlying assumption behind these conditions is that they all refer to entities in the given universe of discourse. For completeness of attributional calculus, constants True and False are also viewed as attributional conditions.

Basic conditions (selectors):

$[x_1 = 1]$, alternatively, x_1	(The value x_1 is 1)
$[x_1 = 0]$, alternatively, $\sim x_1$	(The value x_1 is 0) The alternative notations assume that x_1 is binary.
[color = red]	(The color is red)
[length < 5”]	(The length is smaller than 5 inches)
[temperature ≥ 32° C]	(The temperature is greater than or equal to 32° *C*)
[tools={mallet, knife}]	(The tools are mallet and knife)
[price= ?]	(The price is unknown)
[no_of_eyes = NA]	(The attribute “number of eyes” does not apply)
[patient’s_name = *]	(The patient’s name is irrelevant)

Extended conditions

[color = red v blue v green]	(The color is red, blue or green)
[blood-type ≠ A]	(The blood type is not A)
[length= 4..12]	(The length is between 4 and 12, inclusive)
[color ≠ green]	(The color is not green)
[height > width]	(The height is greater than the width)
[height v width ≤ 3 m]	(The height or the width is smaller than 3 m)
[height & width ≥ 7 cm]	(The height and width are both at least 7 cm)
[height & width < length]	(Both the height and the width are smaller than the length)
[weather = sunny & humid]	(The weather is sunny and humid)
[count(x_1=3, x_3 <13, x_4=2..4) =1]	(Among the conditions x_1=3, x_3 <13, and x_4=2..4, exactly one is true)

The “weather” is a compound attribute. The last example involves the derived attribute “count” (see Section 7.3). Operators “v” and “&” when applied to non-binary attributes or to their values are called *internal disjunction* and *conjunction*, respectively.

As can be seen from the examples above, attributional conditions are easy to interpret and easy to translate into equivalent natural language statements. Basic conditions relate an attribute to one value from the attribute domain. They express conditions on subsets of attribute values, and can be easily generalized by applying *internal disjunction* in the case of nominal attributes, the *range* or *relation* operator in the case of linear attributes, and the *climbing-generalization-tree* operator in the case of structured attributes (Michalski, 1983).

7.2 Syntax and Semantics of Attributional Calculus Sentences

The syntax and semantics of attributional calculus are defined by the following rules specifying well-formed formulas (WFFs); the semantics using the strict interpretation scheme is defined in parentheses).

1. An attributional condition is a WFF.
2. If *S* is a WFF, then ~*S* is a WFF (If *S* is true, then ~*S* is false, and vice versa).

   A negated condition, ~S, can be represented by appropriately changing the relation in the condition. Thus, ~[x = a] is equivalent to [x $\neq$ a], and ~[x > a] is equivalent to [x $\leq$ a].
3. An assignment statement [N := S] is a WFF (N is assigned an expression S).

   Here, N is a derived attribute and S is a WFF or a reference to an externally computed arbitrary function. The symbol ":=" is called the assignment operator. If S is a WFF, then N is a derived attribute whose domain is the same as the domain of the WFF. The domain of N depends on its interpretation. If the WFF is strictly interpreted, then the domain of N is {true, false}. If the WFF is flexibly interpreted, then its domain depends on the method of interpretation (Section 9).

   If S is a reference to an external function, the domain of the function is the domain of N. The assignment statement makes it possible to use arbitrary functions inside of attributional expressions.
4. If *S1* and *S2* are WFFs, then:

S1 & S2	Conjunction: Both *S1* and *S2* are true.
S1 $\vee$ S2	Disjunction: *S1* or *S2* or both are true.
S1 => S2	(alternatively written S2 <= S1) Implication: If *S1* is true then *S2* is true; otherwise, S2 is unknown.
S1 <=> S2	Equivalence: *S1* and *S2* are both either true or false.
S1 $\underline{\vee}$ S2	Exclusive-or: Either *S1* or *S2* is true, but not both.
S1 $\vert\_$ S2	Exception: It can be of two types, strong and weak. The weak exception states that S1 true if S2 is false, otherwise S1 can be true or false. The strong exception states that S1 is true unless S2 is true, in which case S1 is not true. Logically, the weak exception is equivalent to the *or* ($\vee$) operator, and the strong exception to the *exclusive-or* operator ($\underline{\vee}$). Statistically, however, the exception operators differ from the logical *exclusive-or* and *or*. The exception operators are used only when the ratio of the frequency with which S1 is true in the data to the frequency with which S2 is true is high, that is, when it exceeds the *exception threshold*. To distinguish between the weak and the strong exception operators, the first is denoted by "$\vert\_$" and the second by $\Vert\_$.
S1 $\vert^{--}$ S2	S1 is true provided that S2 is also true, otherwise is not. This operator is equivalent to the equivalence operator logically, but not statistically. It is used when the ratio of the frequency with which S1 is true to the frequency with which S2 is untrue is high, that is, exceeds the *provided-that* threshold.

[Count(S1, S2,…Sn) *rel* *R*] The count statement, asserting that the number of true statements in the list (S1, S2, …, Sn) is in relation ***rel*** to ***R***, where ***rel*** and ***R*** are defined as in attributional conditions.

are WFFs.

Note that unlike all other operators, operators |_ and |¯ express not only logical but also *qualitative* statistical information. The form of attributional calculus that uses only basic attributional conditions is called *basic*; if it also uses extended conditions, then it called *extended*.

7.3 Count Expressions

The derived attribute *count* has been introduced to attributional calculus in order to simplify expressions of a class of logic statements that can be very complex and opaque if expressed using classic operators. It represents a special mapping of attributes to a derived attribute that is built-in to the attributional calculus. A general form of the count attribute is:

$$\text{count}(\boldsymbol{S}) \tag{9}$$

where ***S*** is a set of attributional statements, and the "count" stands for the number of statements in ***S*** that are true. Attributional statements that use the count attribute are called *count expressions*. While in the general case ***S*** can contain arbitrary attributional expressions, in the typical use ***S*** contains only attributional conditions. Below is an example of an attributional condition with the count attribute:

$$\text{count}(\text{rev1 = pos, rev2 = pos, rev3 = pos, rev4 =pos}) \geq 3 \tag{10}$$

which evaluates to True if at least three reviews among the four {rev1, rev2, rev3, rev4} are positive.

Let ***X*** be a non-unit subset of attributes with the same domain selected from ***X*****,** the set of all attributes describing an instance. If S consists of attributional conditions involving attributes from ***X***, related by ***rel*** to the same referent ***R***, then one can write:

$$\text{count}\{\boldsymbol{X}\text{:}\ \boldsymbol{REL}\ \boldsymbol{R}\} \tag{11}$$

where ***REL*** is a relation from the set: {EQ, NEQ, GT, EGT, ST, EST}, and ***R*** is defined as in the selector. The symbol EQ means equal, NEQ – not equal, GT – greater than, EGT – equal or greater than, ST- smaller than, and EST – equal or smaller than. These symbols are used in the argument of the count attribute to avoid confusion with the relation in the selector. For example, expression (10) can be alternatively written as:

$$\text{count}(\textit{REVs EQ pos}) \geq 3 \tag{12}$$

where REVs is the set of reviews{rv1, rev2, rev3, rev4}.

As another example, consider the statement

$$\text{count}(\boldsymbol{X}\text{:}\ \textit{ST}\ 5) = 2 \tag{13}$$

This statement says that "the number of attributes from ***X*** that take value smaller than 5 in an instance is equal to 2.

The domain of the count attribute is: {0, 1, 2, …, **max**-1}, where **max** is the cardinality of the set of statements in (9), or of the set of variables in (11).

If the relation ***rel*** holds for all the statements in (9) or all attributes in (11), then one can write

$$\{\boldsymbol{S}: \boldsymbol{REL}\ \boldsymbol{R}\} = all \quad \text{or} \quad \{\boldsymbol{X}: \boldsymbol{REL}\ \boldsymbol{R}\} = all, \text{ respectively} \tag{14}$$

The count attribute is used in the same way as other attributes to create attributional conditions. The count attribute can be used to express an existential and universal quantification over attributional conditions. Assume, for example, that $\boldsymbol{X} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, where all x_i have the same domain, and the predicate EQ(x, v) states that attribute x takes value v.

The existentially quantified expression:

$$\exists\, x \in \boldsymbol{X},\ \ EQ(x, v) \tag{15}$$

can be written in attributional calculus as:

$$\text{count}(\boldsymbol{X}, x\ EQ\ v)\ \geq 1 \tag{16}$$

and expressed in natural language as “the number of variables in X with value v is greater or equal to 1.” The universally quantified expression:

$$\forall x \in\ \boldsymbol{X},\ \ EQ(x,\ v) \tag{17}$$

can be written in attributional calculus as:

$$\text{count}(x\ EQ\ v) = \text{all} \tag{18}$$

In addition to existential and universal quantification, the count attribute allows one to express many other forms of quantification over attributional conditions that do not exist in the conventional predicate logic.

For example:

count(***X***: EQ 4 v 5) > 2 means the number of variables in ***X*** with value 4 or 5 is greater than 2.

count(***X***: GT 5) = 2..4 means the number of variables in ***X*** greater than 5 is between 2 and 4.

count([x=1][y>4], [color=red v blue], [z=2..5]) = 2 means that among:
[x=1][y>4], [color=red v blue], and [z=2..5], two and only two statements are true.

The count expressions are useful in many application domains. For example, it is not uncommon in medicine that a specific disease is indicated when a patient has a number of symptoms greater than some threshold. The *n-of-m* relations are special cases of count expressions. They state that *n* of *m* binary attributes are true in a description (e.g., Sebag, 1999). An *n-of-m* relation is used as a whole description, while in $\mathcal{AC}$, count expressions can be combined with other expressions.

As described above, conditions involving a count operator subsume existential and universal quantifiers over attributional descriptions of entities. For example, the statement: “If two or more conditions from among [x1=2], [x2=4], and [x5 > 1] are true and x6 is between 2 and 6, then d is greater than 4” is expressed as:

$$[\text{count}\{x1 = 2, x2 = 4, x5 > 1\}\ \geq\ 2][x6 = 2..6]\ \Rightarrow [d > 4] \tag{19}$$

The next example illustrates an applicability of count expressions in medicine (courtesy of Dr. P. Koutrovalis). To classify the severity of prostate cancer, three risk factors have been identified:

Factor 1: PSA $\geq$ 10 ng/ml ("PSA" measures the amount of prostate specific antigen)
Factor 2: Gleason's $\geq$ 7 ("Gleason's score" measures the degree of abnormality of cancer cells)
Factor 3: Stage $\geq$ T2b ("Stage" measures the level of disease development)

Using these factors, patients are classified into four general categories representing increasing severity of the disease: Category 1-No factors present, Category 2-One factor present, Category 3-Two factors present, and Category 4-All three factors present. These categories can be described formally in attributional calculus in a straightforward manner:

count(PSA $\geq$ 10 ng/ml, Gleason's $\geq$ 7, Stage $\geq$ T2b) = 0 $\Rightarrow$ category = 1
count(PSA $\geq$ 10 ng/ml, Gleason's $\geq$ 7, Stage $\geq$ T2b) = 1 $\Rightarrow$ category = 2
count(PSA $\geq$ 10 ng/ml, Gleason's $\geq$ 7, Stage $\geq$ T2b) = 2 $\Rightarrow$ category = 3
count(PSA $\geq$ 10 ng/ml, Gleason's $\geq$ 7, Stage $\geq$ T2b) = 3 $\Rightarrow$ category = 4

The count expressions are also useful for simply expressing symmetric Boolean logic functions. Such functions tend to have the longest expressions in terms of standard logic operators (Michalski, 1969). By using a count attribute, expressions of these functions are very simple.

7.4 Other Standard Derived Attributes

In addition to count attributes, $\mathcal{AC}$ also recognizes other standard derived attributes. Let x $\in$ ***X***, $\mathcal{D}$(x) be the domain of x, and ***X*** $\subseteq$ $\mathcal{X}$, where $\mathcal{X}$ is the set of all attributes in an instance (event). Here are standard derived attributes:

|x| expresses the number of values in $\mathcal{D}$(x), if $\mathcal{D}$(x) is finite. If $\mathcal{D}$(x) is infinite, then |x| = ∞.

x^+ and x^- express, respectively, the next and the previous value of x in $\mathcal{D}$, if $\mathcal{D}$ is totally or cyclically ordered. If $\mathcal{D}$ is a hierarchically structured domain, then x^+ is the parent of the value x, and x^- is a child of x. Thus, if selector x^- = A is satisfied, if any child of x is A. If $\mathcal{D}$ is an unordered set (x is a nominal attribute), then x^+ and x^- are elements in $\mathcal{D}$ different from x. Thus, selector x^+ = A is satisfied if x $\neq$ A.

min_x and max_x express, respectively, the minimum and the maximum values of x in $\mathcal{D}$.

max_***X***, min_***X***, and avg_***X*** express the maximum, minimum and average, respectively, of values of attributes defined in ***X*** in an instance. Attributes listed in ***X*** must have the same linearly ordered domain. In the case of attribute avg_***X***, the domain the attributes in ***X*** must be metric. For example, if a concept instance is:

(x_1=red & x_2=3 & x_3=6 & x_4=4 & x_5=tall & x_6=9 & x_7 =8, and ***X*** = {x_2, x_3, x_4, x_6, x_7)

then max_***X*** =9, min_***X*** =3 and avg_***X*** = 6.

sum_***X*** and prod_***X*** express the arithmetic sum and the product, respectively, of attributes in ***X***. These derived attributes also apply only to metric attributes.

diff_***X*** and ratio_***X***, where ***X* =** {x_i, x_j}, and x_i, x_j are two attributes selected from ***X***, measure the difference ($x_i – x_j$), and the ratio (x_i / x_j). The diff_***X*** attribute applies to attributes of type interval, ratio and absolute. The ratio_***X*** applies to attributes of ratio and absolute types.

A question arises as to how to determine the set ***X***. Determining ***X*** that yields highly relevant new derived attributes is a problem by itself. A search for the "best" ***X*** may be done by using heuristics drawn from the domain knowledge or by experimentation. Because ***X*** must satisfy restrictions as to the types of attributes in it, and typically will be rather small, the "best" ***X*** can be determined by an exhaustive search.

The reasons for introducing the above derived attributes to $\mathcal{AC}$ are that they may lead to both simple and understandable inductive hypotheses that, if expressed using conventional logic operators, would be very complex and difficult to understand. The book by Liu and Motoda (1998) provides a review of methods concerned with construction of attributes.

7.5 Important Cases of Attributional Descriptions

Attributional descriptions are ordered sets of attributional calculus expressions that describe an entity, and set of entities, or relations among entities in terms of their attributes. To make this definition sufficiently general, the term "entity" includes here any objects, as well as concepts, decisions or classes assigned to these objects. The attributes may be directly measurable on the entities, or derived from the measurable ones. This section defines selected forms of attributional descriptions that are of particular importance for natural induction.

A conjunction of attributional conditions is called a *conjunctive description,* or a *complex*.

For example, an attributional description of my "favorite house" may be:

[Loc=in _ forest] & [Color=white v brown] & [#Bedrooms=4..5] & [School_quality ≥ good] &
[Backyard=large&private] & [Distance-to-work-H & Distance-to-work-W ≤ 5miles] &
[LocDrgDlrs=prsn] & [Shops-work ∨ Shops-home ≤ 3miles] & [count(shops) >1] (20)

The attribute "Backyard" is compound, with constituent attributes "size" and "privacy_level". The attributes "Distance-to-work-H" and "Distance-to-work-W" measure the distance from home to workplace for the husband and for the wife, respectively. The last condition means: "The location of drug dealers who operated in the local area is prison."

A disjunction of complexes is called a *disjunctive description* or *disjunctive normal form* of attributional calculus. As known from formal logic, a disjunctive description can describe any subset of the event space, so it can characterize every possible concept in terms of the attributes spanning the event space.

An expression:

Premise* => *Consequent (21)

where ***Premise (***also called ***Condition)*** and ***Consequent*** (also called ***Decision*** or ***Action***) are conjunctive descriptions, is called an *attributional rule*. An attributional rule can also be written in the opposite direction of implication, that is,

Consequent <= Premise (22)

Here is an example of an attributional rule:

[Weather=warm & sunny] & [Have_partner] => [Play = tennis]

The term “weather” is used the example as a compound attribute, the term “Have_partner” as a binary attribute, and the term “Play” as a multi-valued attribute, whose values are different games one can play.

An attributional rule in which all conditions (selectors) are basic is called *basic*. If a premise in a rule (21 or 22) includes one or more conditions that themselves are attributional rules, then such a rule is called an *implicative attributional rule.*

To distinguish between the “external” implication that links the rule premise with the rule consequent, and the “internal” implication within the rule premise, the external implication is denoted by =>, and the “internal” implication by ->. Note that the internal implication is not a logical implication in the classic sense, but rather a strong connection, indicating that if the left side is satisfied, the right side must be as well for the rule to fire.

Here is an example of a simple implicative rule:

[Have_partner] & (weather=warm -> outdoor-court=available) &
(weather=cold -> indoor-court=available) => [Play=tennis]

Implicative rules can succinctly represent a description that otherwise would be more complicated and less understandable. Consider, for example, the problem of learning concept, C, represented by the positive and negative examples in Figure 8.

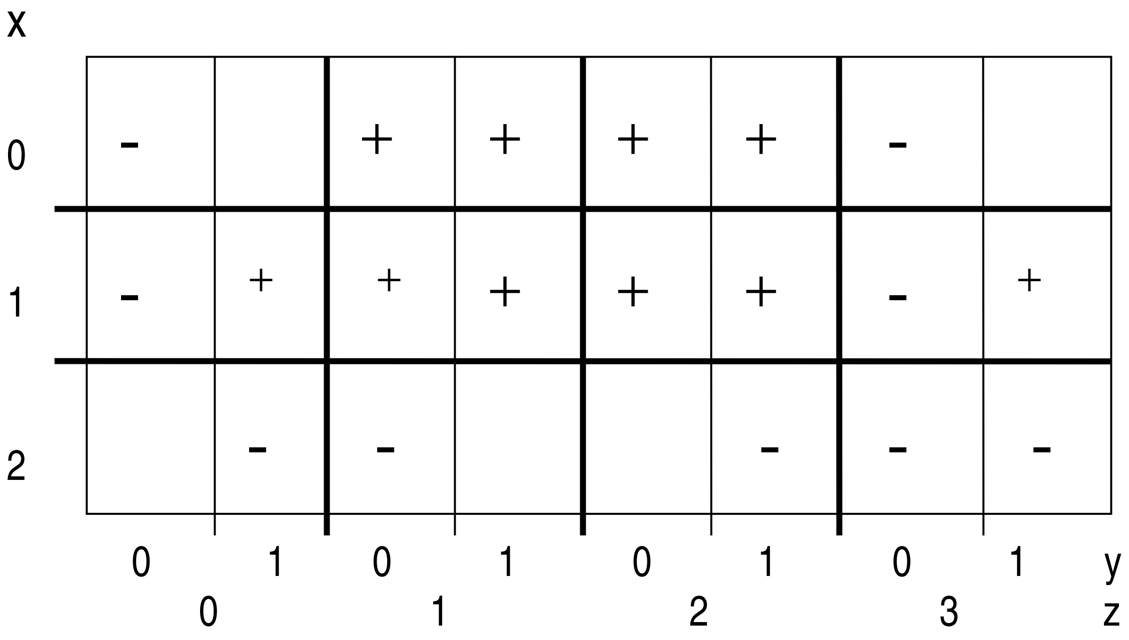


Figure 8: Training data used for illustrating an implicative rule.

To describe concept, C, one needs five basic decision rules:

x=0 & z=1 => C;
x=0 & z=2 => C;

x=1 & z=1 => C;
x=1 & z=2 => C;
x=1 & y=1 => C

two standard attributional rules:
x = 0 v 1 & z = 1 v 2 => C;
x = 1 & y =1 => C

and just one implicative attributional rule:
x = 0 v 1 & (z = 3 -> y = 1) => C

An expression:

Premise => Consequent |_ Exception (23)

where ***Premise, Consequent*** and ***Exception*** are complexes, is called a *censored attributional rule* (or a *rule with an exception clause*). Here |_ denotes the weak exception operator, and such a censored rule is read, "If ***Premise*** is true, then ***Consequent*** is true, except when ***Exception*** is true, in which case ***Consequent*** may or may not be true."

This *weak* form of a censored rule is different from the *strong* form, introduced in (Michalski and Winston, 1986), which assumes the strong exception operator. In a strong censored rule, ***Consequent*** is false when ***Exception*** is true. The reason for using the weak exception operator in (23) is that weak censored rules are much simpler to implement, and learning and applying them is much easier.

Censored rules can also be written in the opposite direction of implication, that is,

Consequent <= Premise |_ Exception (24)

Here is an example of a censored rule:

[Weather=nice] & [Have_partner] => [Play = tennis] |_ *[Class-CLD873 = still-to-prepare]*

From the logical viewpoint, a weak censored rule can be re-written into:

Premise & ~Exception => Consequent (25)

and a strong censored rule can be re-written into

Premise & ~Exception => Consequent
Premise & Exception => ~Consequent (26)

Censored rules have, however, also statistical aspect, that is, are used only for describing situations in which a rule: ***Premise => Consequent*** holds very frequently (represents a strong pattern), but in some cases, defined by the ***Exception***, does not hold. Specifically, a censored rule is used when the ratio of the frequency of satisfying the ***Premise*** to the frequency of satisfying the ***Exception*** is greater than the *exception threshold.* Figure 9 illustrates an advantage of using censored rules.

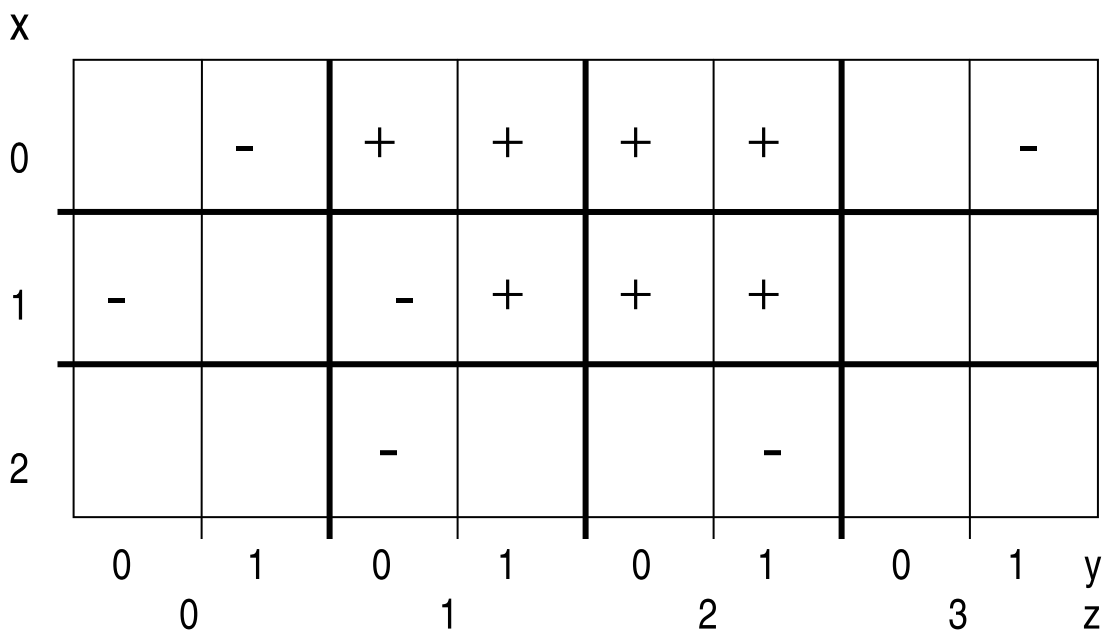


Figure 9: A training set for illustrating censored attributional rules.

To describe concept C represented by positive examples, three standard $\mathcal{A}C$ rules are needed:

x=0v1 & z=1 & y=1 => C; x=0 & z= 1 => C; x=0v1 & z=2 => C

Using the exception operator, only one censored rule is needed:

x=0v1 & z =1 v 2 => C |_ x=1 & z=1 & y=0

An expression:

Premise => Consequent* |¯ *Precondition (27)

where **Premise, Consequent** and **Precondition** are complexes, is called a *preconditioned attributional rule* (or a *rule with provided-that clause*). Analogously to weak and strong censored rules, there are also weak and strong preconditioned rules. A weak preconditioned rule, as shown in (27), is interpreted, "If ***Premise*** is true, then ***Consequent*** is true, provided that ***Precondition*** is true; if ***Precondition*** is false, then the truth-status of ***Consequent*** is unknown. In the case of a strong preconditioned rule, if ***Premise*** is true and ***Precondition*** is false, then ***Consequent*** is false as well. The reason for using the weak precondition operator in (27) is that weak preconditioned rules are much simpler to implement, and learning and applying them is much easier.

A preconditioned rule can also be written in the opposite direction of implication, that is, ***Consequent* <= *Premise*** |¯ ***Precondition***.

From the logical viewpoint, a preconditioned rule can be re-written into:

Precondition & Premise => Consequent (28)

A strong preconditioned rule can be re-written as two rules:

Precondition & Premise => Consequent (29)

~Precondition & Premise => ~Consequent (30)

Here is an example of a (weak) preconditioned rule:

[Weather=warm & sunny] & [Have_partner] => [Play=tennis] |¯¯ *[Class-CLD873=prepared]*

As do censored rules, preconditioned rules have also a statistical property. They are used only when the ratio of the frequency of satisfying the ***Precondition*** to the frequency of satisfying the ***Premise*** is below a *precondition rule threshold*.

If a set of attributional rules (standard, censored or preconditioned) all have the same ***Consequent***, it is then called a *ruleset*. If the ***Consequent*** in a ruleset states a concept name, the ruleset is called a *concept description*. A ruleset can be transformed into a logically equivalent single implicative statement whose Premise is a disjunction of the Premises of the rules in the ruleset. The precondition operator can also be applied to a ruleset, which case we have a *preconditioned ruleset*:

{Premise => Consequent} |¯¯ ***Precondition*** (31)

A preconditioned ruleset provides a mechanism for expressing a concept description which consists of attributional rules that have one or more common conditions.

An attributional rule in which the ***Consequent*** defines a decision or action to be assigned to a situation satisfying the ***Premise*** is called a *decision rule*. For example, [price=low] & [quality=satisfactory] => [decision=buy].

An attributional rule in which the ***Consequent*** is a conjunction of two or more attributional condition is called a *multihead rule*; otherwise it is called a *single-head rule*. Multihead rules characterize situations in which the truth of the ***Premise*** implies the truth of several consequent conditions.

A collection of rulesets is called a *classifier* (or a *ruleset family*) if the ***Consequent*** of the rules in the ruleset spans all the values in the domain of an output attribute (all the concepts in a set of concepts to be learned). For example, if the domain of an output attribute is a set of diseases, then a collection of rulesets for all the diseases in the set is a classifier.

A collection of ruleset families for two or more decision attributes from the same database, with associated background knowledge (definitions of attribute types, rulesets parameters, and related information) and an associated data (e.g., the training and testing sets) is called a *knowledge system.*

7.6 Cognitive Constraints

In standard formal logics, such as propositional calculus or predicate logic, there is no constraint regarding the length of expressions or the number of times any logic operator can be repeated in an expression. Because the main objective of attributional calculus is to provide a representational system to support natural induction, some constraints need to be introduced. Clearly, natural language sentences need not only to be grammatical, but also to satisfy some constraints on their length and structure if they are to be understood.

To this end, the definition of attributional calculus includes, in additional to syntax and semantics, also constraints on the complexity of its descriptions, called *cognitive constraints*. These constraints reflect preferences and limitations imposed on the descriptions in order to make the logically natural language expressions appear natural to people, and easy to interpret and understand. A precise definition of such constraints cannot be formulated because different people have different cognitive capabilities, thus such constraints would need to be tailored to each individual. Therefore, this section provides only general ideas on how such constraints can be introduced into attributional calculus in order to implement natural induction.

Cognitive constraints are defined by parameters limiting the number of repetitions of different operators in a single attributional expression. These parameters include:

A. *Conjunction_limit*, which defines the maximum number of conditions allowed in a conjunctive description. For example, *Conjunction_limit* = 5.

B. *Internal_conjunction_limit*, which defines the maximum allowed number of items (variables or values) joined by an internal conjunction in an attributional condition. A reasonable constraint may be, for example, *Internal_conjunction_limit* = 4.

C. *Disjunction_limit*, which defines the maximum number of conjunctive descriptions allowed in a single disjunctive description. For example, such a constraint may be *Disjunction_limit* = 3.

D. *Internal_disjunction_limit*, which defines the maximum number of attributes or values that can be linked by internal disjunction in an attributional condition. For example, such a constraint may be *Internal_disjunction_limit* = 4.

E. *Implication_limit,* which defines the maximum number of statements that can be linked by an implication operator in a single expression. For example, if *Implications_limit* = 2, expressions will be limited to attributional rules.

F. *Equivalence_limit* defines the maximum number of statements that can be linked by an equivalence operator in a single expression. For example, such a constraint may be *Equivalences_limit* = 2.

G. *Exception_limit,* which defines the maximum number of statements that can be linked by an exception operator.

The above constraints need to be interdependent. If one constraint is satisfied by a significant margin, another constraint may be relaxed, and the other way around. Also, it is known from research in cognitive science that humans have preferences regarding different operators. For example, people usually prefer a conjunctive description over a disjunctive one. They also prefer statements asserting some property than negating it. For example, the condition *color = red* is viewed as cognitively simpler than *color* $\neq$ *red*. Such preferences can be modeled by assigning different *conceptual complexity* to different operators.

To reflect the above considerations, there may also be a constraint imposed on the total complexity of a single expression. The complexity of an expression is defined by a function:

$$\begin{aligned}\text{Complexity} = {} & a * \text{Conjunction\_count} + b * \text{Internal\_Conjunction\_count} + \\ & c * \text{Disjunction\_count} + d * \text{Internal\_Disjunction\_count} + \\ & e * \text{Implication\_count} + f * \text{Equivalence\_count} + g * \text{Exception\_count}\end{aligned} \tag{32}$$

where Conjunction_count, Internal_Conjunction_count, …, Exception_count denote the number of statements connected by conjunction, internal conjunction, …, exception operators, respectively, and coefficients *a*, *b*, *c*, *d*, *e*, *f* and *g* reflect estimates on the "cognitive cost" of these individual operators in an expression. Such estimates can be determined experimentally to fit the preferences of given users.

The above cognitive constraints are imposed on the results of inductive inference (descriptions, hypotheses, rules, patterns). An expression that satisfies the cognitive constraints is called *admissible*, otherwise, it is called *inadmissible.* An inadmissible description can be transformed to a logically equivalent set of admissible descriptions by applying the *name assignment* transformation rule:

Any admissible description within an inadmissible description can be replaced by a name (binary attribute) assigned to it. The name and the admissible description to which it is assigned are then added to the body of the attributional description.

Let us explain the application of this rule to an inadmissible expression. An attributional description was defined as an ordered set of attributional expressions. When replacing a part of an inadmissible expression by its name, it is assumed that a pair consisting of the name and the expression assigned to it will follow the expression in which the name is used.

For example, let us assume *Conjunction_limit* = 5, and that we have an inadmissible conjunctive description:

$$A \,\&\, B \,\&\, C \,\&\, D \,\&\, E \,\&\, F \,\&\, G \,\&\, H \,\&\, I \tag{33}$$

where A, B, ..., I are attributional conditions.

Applying the name assignment transformation rules, the expression (33) can be transformed into the following admissible description:

$$A \,\&\, B \,\&\, C \,\&\, D \,\&\, J,\ \ J := E \,\&\, F \,\&\, G \,\&\, H \,\&\, I \tag{34}$$

The description (34) is read:

"A and B and C and D and J, where J is E and F and G and H and I."

Another transformation would yield:

$$<A \,\&\, B \,\&\, C \,\&\, J \,\&\, K,\ J := D \,\&\, E \,\&\, F;\ \ K := G \,\&\, H \,\&\, I > \tag{35}$$

As one can see, a given inadmissible description can be transformed to many different admissible descriptions. An advisable strategy is to seek an admissible description with the minimum number of expressions. In a similar fashion, one can transform an inadmissible description using other operators into an admissible one.

8 EXPRESSIVE POWER OF ATTRIBUTIONAL RULES

Let us now characterize the class of functions that can be described by attributional calculus. Let $x_1, x_2, \ldots, x_n$, be input attributes, and $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ be their domains, respectively; let $y_1, y_2, \ldots, y_m$ be output attributes, and $\mathcal{D}^1, \mathcal{D}^2, \mathcal{D}^3, \ldots, \mathcal{D}^m$ be their domains, respectively. Let us first assume that all input and output attributes are discrete and finite.

Theorem 1

Every mapping:

$$f: \quad \mathcal{D}_1 \text{ x } \mathcal{D}_2 \text{ x } \mathcal{D}_3 \text{ x } \ldots \text{ x } \mathcal{D}_n \text{ ----> } \mathcal{D}^1 \text{ x } \mathcal{D}^2 \text{ x } \mathcal{D}^3 \text{ x } \ldots \text{ x } \mathcal{D}^m \tag{36}$$

where x denotes the Cartesian product, can be expressed by a set of attributional rules, specifically, by a family of rulesets.

Proof

Since output domains are discrete, the Cartesian product $\mathcal{D}^1$ x $\mathcal{D}^2$ x $\mathcal{D}^3$ x … x $\mathcal{D}^m$ is also discrete. Assume that $\mathcal{D}^o = \mathcal{D}^1$ x $\mathcal{D}^2$ x $\mathcal{D}^3$ x ….x $\mathcal{D}^m$. The function (36) can then be represented as:

$$f: \quad \mathcal{D}_1 \text{ x } \mathcal{D}_2 \text{ x } \mathcal{D}_3 \text{ x } \ldots \text{ x } \mathcal{D}_n \text{ ----> } \mathcal{D}^o \tag{37}$$

$\mathcal{D}_1$ x $\mathcal{D}_2$ x $\mathcal{D}_3$ x … x $\mathcal{D}_n$ defines a finite instance space, which is partitioned by the function f into a finite number, $|\mathcal{D}^o|$, subspaces. Each subspace corresponds to one value of $\mathcal{D}^o$, and can be described by an attributional ruleset that maps that subset into that value. A family of rulesets representing each value from $\mathcal{D}^o$ represents function (37). QED

Theorem 2

Every mapping:

$$f: \quad \mathcal{D}_1 \text{ x } \mathcal{D}_2 \text{ x } \mathcal{D}_3 \text{ x } \ldots \text{ x } \mathcal{D}_n \text{ ---> } \{0,1\} \tag{38}$$

can be represented by a single implicative attributional rule. As shown before, such a mapping can be represented by an attributional ruleset. If at least one rule in this set is satisfied by an event, then the value "1" is outputted, otherwise the value "0". A set of attributional rules with the same consequent is logically equivalent to a disjunctive normal form (DNF), that is, a disjunction of premises (complexes) of the rules from the ruleset. It is known from logic that for every DNF expression there exists a logically equivalent conjunctive normal form (CNF), that is, a conjunction of disjunctions of selectors. Consider one such disjunction:

$$(S_1 \vee S_2 \vee S_3 \vee \ldots S_k) \tag{39}$$

where $S_1, S_2, S_3, \ldots, S_k$ are selectors. Using the equivalence (A => B) <=> (~A $\vee$ B), and de Morgan's Law, ~(A $\vee$ B) <=> ~(A&B), the expression (31) can be transformed into:

$$(\sim S_1 \sim S_2 \sim S_3 \ldots \text{ -> } S_k) \tag{40}$$

By transforming all disjunctions in the CNF to implications (40), we get a single attributional implicative rule. QED

Let us now assume that some input variables are continuous, that is, that some domains $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, ..., $\mathcal{D}_n$ are infinite. In this case, the set $\mathcal{D}_1$ x $\mathcal{D}_2$ x $\mathcal{D}_3$ x $\mathcal{D}_n$ is infinite. The function (38) cannot be exactly represented by a family of attributional rulesets, but can be approximated by it. A straightforward way to do this is to discretize all continuous attributes, so that their domains will be finite, which will bring us to the previous case.

Such a discretization of continuous attributes can be avoided if the task is to learn a description of the function (38) from a finite number of instances. If the number of instances is finite, then one can create a consistent and complete generalization of them in the form of a family of attributional rulesets. This can be done by mapping the finite set of instances into subspaces whose boundaries are specified by ranges of attribute values. Such ranges are represented in $\mathcal{AC}$ using the range operator "..", defined in attributional calculus.

Due to a rich repertoire of easy to interpret forms and operators, $\mathcal{AC}$ can not only represent any function (28), but represent it concisely and understandably. Each such representation can be directly translated to natural language.

9 $\mathcal{AC}$ INTERPRETATION SCHEMAS

The previous sections discussed attributional calculus with a strict (binary) interpretation schema, which treats $\mathcal{AC}$ expressions as either true or false. Specifically, it evaluates them to a value from the set {0, 1}, where "1" stands for "True" and "0" for False. While such an interpretation has advantages, in some situations it is preferable to evaluate them flexibly, as multi-valued or continuously-valued expressions (descriptions). Instead of declaring that an expression is true or false, a flexible evaluation assigns to it a degree of truth, or a degree of match between the description and the event. A flexible interpretation can be particularly advantageous when $\mathcal{AC}$ expressions are used as concept descriptions. For example, if a concept description is an attributional conjunction characterizing the concept, then its flexible interpretation is related to a model of human concept representation (Medin, Wattenmaker, and Michalski, 1988).

Let us start with a definition of an *$\mathcal{AC}$ interpretation schema*. Given an event e (an example, an instance, or a model) and an $\mathcal{AC}$ description D, an interpretation schema produces a degree of match, $DM(e, D) \in [0,1]$, between the event and the description. If an interpretation schema evaluates expressions solely into the end-points of the interval [0,1], then it is called *strict*; if it evaluates expressions into the whole interval, it is called *flexible*. An interpretation schema is a combination of methods for interpreting $\mathcal{AC}$ selectors and operators.

This section presents several alternative $\mathcal{AC}$ interpretation schemas. Because the primary goal of this paper is to present attributional calculus as a tool for natural induction, we focus here on interpretation schemas for $\mathcal{AC}$ expressions that are of primary importance to this application, specifically, selectors, rules, rulesets, and ruleset families (classifiers). The described schemas define methods for determining a degree of match between an event and a concept description. To classify an event, a degree of match between the event and all candidate concept descriptions is determined, and the highest degree of match indicates the concept. When a strict match schema is used, it is sufficient to determine only the description that strictly matches the event.

When a concept description is in the form of an ordered family of rulesets, rulesets are evaluated sequentially, and the first ruleset that is matched indicates the concept.

A question arises as to what to do when two or more concept descriptions are matched with the same or similar degrees. In the AQ learning methodology, such concept descriptions are output as alternative concept descriptions (Michalski and Kaufman, 2001).

Let: e be an example or instance to be matched against a ruleset

RS be a ruleset, and R_1 $R_{1,}$ R_2 , …, R_m be individual rules in this set

FRS be a classifier, defined by a family of rulesets RS_1, RS_2 , …, RS_r

P be a premise (conjunctive expression) in a rule

S be a selector (condition) in a rule

C be a consequent in an attributional rule, for example, an assignment of a concept or a decision class to the given output variable

p, *n* denote the total number of positive and negative examples, respectively, that are covered by a single rule, R_i, in a ruleset

P, *N* denote the total number of positive and negative examples, respectively, in the training set for a concept

As mentioned above, to assign a concept (generally, a rule's consequent) to an event, one needs to determine the "best" match between the event and the rulesets in the classifier. When using a strict interpretation schema, the consequent of the ruleset with the match of 1 (perfect match) is assigned to the event, without the need for evaluating other rulesets. This feature is an important advantage of a strict interpretation schema. To apply such a method, the descriptions of different concepts must be logically disjoint (the case in which they intersect is considered later).

When a flexible interpretation schema is used, degrees of match between the event and all rulesets in the classifier need to be computed. The event is assigned the concept indicated by the ruleset with the highest degree of match. In some applications, a different cost is assigned to different types of errors or misclassification. For example, the cost of not recognizing an early stage of cancer is much greater than the cost of erroneously diagnosing the cancer. In such applications, the minimum misclassification cost should be used as a criterion for assigning a decision.

Let us assume that we want to determine a degree of match, DM(e, RS), between an example, e, and an attributional ruleset, RS, that is one of the rulesets in a classifier. To do so, a rule interpretation schema needs to know the degree of match, DM(e, R_i), between the event e and each rule in RS, and then to aggregate these degrees into one DM(e, RS) degree of match. To know DM(e, R_i), the schema has to compute degrees of match, DM(e, S_j) between the event, e, and selectors, S_j, in each rule R_i, and then aggregate them into one value.

An $\mathcal{AC}$ interpretation schema must thus define methods for determining the following degrees of match: (1) DM(e,S), between an event, e, and selector, S, in a rule, (2) DM(e, R), between an event, e, and rule R, and (3) DM(e, RS), between an event, e, and ruleset, RS. It must also define a method ASSIGN(e, FRS) for assigning a concept (generally, asserting a rule's consequent) to the example e using classifier FRS.

Let us now consider different methods for computing the above degrees.

Selector Interpretation Methods: Methods for computing DM(e, S)

1. *CSI* ("Strict selector interpretation")

   DM(e, S) = 1 if the event e matches selector S strictly, as described in Section 7.1), otherwise 0. This method can be applied to all types of attributes. Figure 10 illustrates the CSI method for a selector in which a metric attribute is assigned the range of values: x = a..b.

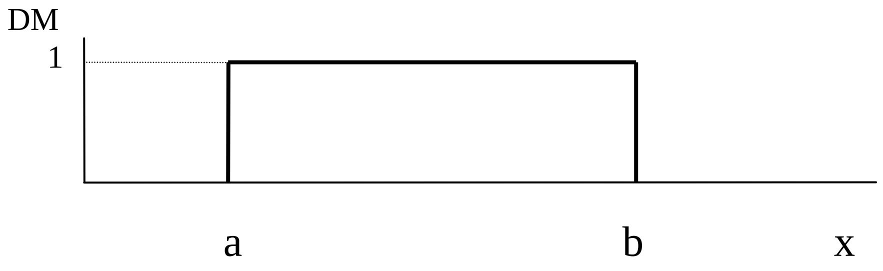


Figure 10: An illustration of a strict method for selector interpretation.

2. *FSI* ("Flexible selector interpretation") ***Default method***

   For nominal attributes, FSI works the same way as the strict method. For metric attributes, instead of sharp borders of attribute ranges in selectors, more gradual ("fuzzy") boundaries are used. This is so because in many applications, sharp borders are not justified. For example, if the likely time of occurrence of a plant disease is described by a selector [time = August..September], it would be unreasonable to reject the possibility of this disease being detected on July 31.

   There can be many ways of smoothing ("fuzzyfying") sharp borders of attribute ranges in selectors. A computationally simple way (used as default in $\mathcal{AC}$) is to assume that the degree of match, DM, decreases linearly with the distance from the end point(s) of a range. If a selector defines an inequality relation, e.g., $x \leq a$, only one endpoint (here, "a") needs to be made gradual, because the other one is the end of the attribute domain. To consider a general case, assume that S is [x = a..b], where a, b are not the extreme values of the attribute domain.

   Then DM(e, S) = 1 if the attribute in question in event e takes a value between a and b, inclusive; otherwise the degree of match is f(d), where f is a monotonically decreasing function of d, the distance of the value of the attribute in e to the nearest the endpoint (a or b). As a default, f is assumed to be a linear function whose slope, controlled by a parameter k, depends on the width of the range. Specifically, the slope is $1/\delta$, where $\delta = |b - a| / k$ (Figure 11).

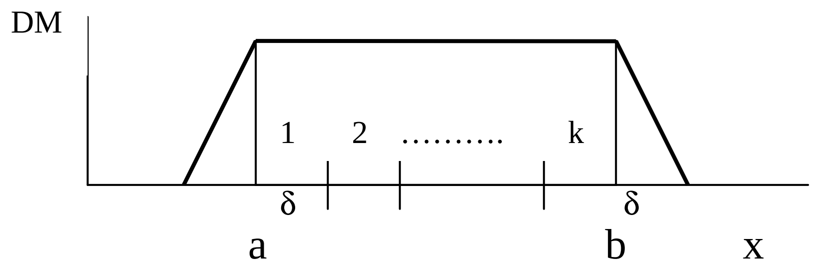


Figure 11: An illustration of the degree of match function for the selector x=a..b, defined by a flexible interpretation schema.

Rule Interpretation Methods:

1. SC ("Standard conjunction")

   DM(e, R) = 1 if all selectors in R are matched strictly, that is, for all selectors, Si, in the premise of rule R, DM(e, Si) = 1; otherwise DM(e, R) = 0.

2. MIN (Minimum of selector degrees of match)

   DM(e, R) = Min(DM(e, Si)) over the selectors Si in the premise of rule R. The method is applicable with a flexible selector evaluation schema.

3. MIN-w (Minimum of weighted selector degrees of match)

   In this method, each selector in the rule's premise is assigned a weight p / (p+n), where p is the number of positive training events satisfying the selector, and n is the number of negative events satisfying the selector. Although the method can be applied with a strict or flexible selector interpretation method, the flexible method is assumed by default. When a flexible selector interpretation method is used, the selectors' degree of match is multiplied by the selector's weight.

   DM(e, R) is the minimum of the weighted degrees of selector matches by the selector weights.

   This method is more computationally costly than MIN, because determining all of the p and n values may be computationally costly if the training dataset is very large.

4. PROD (Product of selector degrees of match)

   DM(e, R) = $\prod$(DM(e, Si) over the selectors Si in rule R. The method is applicable to a flexible selector interpretation schema.

5. AVG (Average of degrees of selector match)

   DM(e, R) is the average of the degrees of match, DM(e, Si), between event e and selectors Si in the rule. If all selectors are evaluated strictly, DM(e, R) is the ratio of the number of selectors satisfied by the event e to the total number of selectors in the premise of R.

6. AVG-W (Average of weighted selector degrees of match)

   This method is similar to AVG, but it multiplies the degrees of selector match by selector weights (defined as p / (p+n)), analogously to the MIN-W schema.

   DM(e, R) is the average of the degrees of selector matches multiplied by the selector weights.

   This method is applied with a flexible selector interpretation.

Ruleset Interpretation Methods

1. SD ("Standard Disjunction")

   This method interprets a ruleset as a logical disjunction of the premises of the rules in the ruleset. If the premise of at least one rule in ruleset RS is strictly satisfied by event e, then DM(e, RS) =1, otherwise, it is 0.

2. MAX ("Maximum of the degrees of rule match")

   This method interprets a ruleset to the maximum of the degrees of match of rule premises, that is, DM(e, RS) = MAX(DM(e, Ri)

3. PSUM ("Probabilistic sum of the rules' degrees of match")

   This method interprets a ruleset to the probabilistic sum of the degrees of match of its rules, that is, DM(e, RS) = PSUM(DM(e, Ri)). The PSUM is computed iteratively according to the rule: PSUM(d1, d2) = d1 + d2 – d1d2.

4. AVG ("Average of the degrees of the rule match")

   This method interprets a ruleset into the average of the degrees of match of its rules, that is, DM(e, RS) = AVG(DM(e, Ri).

As mentioned earlier, an $\mathcal{AC}$ interpretation schema is defined by a combination of methods for selector, rule, and a ruleset interpretation. In the above, we described two selector interpretation methods, six rule interpretation methods, and four ruleset interpretation methods; thus, there are potentially 48 $\mathcal{AC}$ evaluation schemas.

Some of these combinations amount to logically equivalent $\mathcal{AC}$ schemas. For example, when a strict selector interpretation method is used, then the SC (standard conjunction) and the MIN methods for rule interpretation are equivalent. When SC is used for rule interpretation, then the SD (standard disjunction) and the MAX methods for ruleset interpretation are equivalent.

Below are four basic $\mathcal{AC}$ interpretation schemas. The first (default), third and fourth are flexible, and the second is strict.

1. Basic Flexible Interpretation Schema (BFIS)—used as default for $\mathcal{AC}$.

BFIS is defined by the following combination of methods:

Selector interpretation:	FSI	(Flexible selector interpretation)
Rule interpretation:	MIN	(Minimum of the selector degrees of match)

Ruleset interpretation	MAX	(Maximum of the rule degrees of match)

2. Strict Interpretation Schema (SIS)

SIS interprets attributional calculus expressions as binary logic expressions (as in Section 7.2). It is defined by the following combination of methods:

Selector interpretation:	CSI	(Strict selector interpretation)
Rule interpretation:	SC	(Standard conjunction)
Ruleset interpretation:	SD	(Standard disjunction)

3. Weighted Interpretation Schema (WIS)

WIS is defined by following combination of methods:

Selector interpretation:	FSI	(Flexible selector interpretation)
Rule interpretation:	MINW	(Minimum of weighed selector degrees of match)
Ruleset interpretation	MAX	(Maximum of degrees of matches)

4. Probabilistic Interpretation Schema (PIS)

PES is defined by following combination of methods:

Selector interpretation:	FSI	(Flexible selector interpretation)
Rule interpretation:	PROD	(Product of selector degrees of match)
Ruleset interpretation:	PSUM	(Probabilistic sum of degrees of rule matches)

Given an event, e, and a ruleset, RS, an $\mathcal{AC}$ interpretation schema generates a degree of match, DM(e, RS), between the event, e, and the ruleset, RS. When the event is to be assigned a concept (or a decision class, a value of an output attribute, or generally, a rule consequent) from a set of candidate concepts, the degrees of match between the event and the rulesets for all candidate concepts (defined in the family of rulesets, FRS) are computed, and used for making such an assignment.

Let us define methods for making this assignment, that is, methods that given an event, e, and a classifier FRS, assigns to it a rule consequent.

1. MaxMatch (Maximal degree of match)

MaxMatch(e, FRS) assigns to e the consequent of a ruleset, RS $\in$ FRS, for which the degree of match, DM(e, RS) is maximum.

If the ruleset interpretation schema is strict (CSI) and rulesets in FRS are logically disjoint, then the consequent of the first ruleset that matches strictly the event is selected for output. There is no need for matching the event against other rulesets.

If more than one ruleset matches the event with a similar degree of match, then there are two ways to handle this:

(1) The example is assigned the concept that has the highest prior probability among the alternative concepts, based on the prior frequency of concepts (decisions, in general, consequents).

(2) The example is assigned multiple concepts, the concepts specified by the equally matched rules. Such an assignment is the most appropriate in two cases:

a) The output variable is set-valued. The matched rulesets generate a set of values to be assigned to the output variable. Such a situation exists, for example, when a value of a set-valued variable represents a tool that can be applied to a given problem, and a set of such values represents all tools needed to do the job.

b) The output variable is single-valued, but one does not want to force a single decision on an example that similarly matches different rulesets, but prefers to assign to it a list of alternative assignments. Such assignments can then be reduced to one by obtaining more information about the example and using more specific rulesets. This latter method is fairly typical in human decision making.

2. MinCost(e, FRS), assigns to e the consequent of a ruleset in FRS that represents the minimum error cost. This method is used when different costs are assigned to different decisions.

10 ANNOTATED ATTRIBUTIONAL CALCULUS

Previous sections described a bare form of attributional calculus, which consists solely of logic-style expressions. When learning concept descriptions from examples, it is useful to annotate the induced hypotheses in terms of numerical parameters characterizing them. Because natural induction is concerned with learning attributional rules and ruleset families, we will focus our attention to such expressions.

An *annotated attributional rule* has been defined as:

$$\textit{\textbf{Premise}} \;\Rightarrow \textit{\textbf{Consequent: Annotation}} \qquad (41)$$

where ***Premise*** and ***Consequent*** are *complexes* (conjunctions of attributional conditions), and ***Annotation*** is a list of parameters characterizing the rule and possibly other relevant information. There can be many useful parameters characterizing attributional rules and attributional rulesets. Here we will limit ourselves to several basic ones. The basic parameters are:

Support *(*or *positive coverage*) defined as the probability, ***p(Premise | Consequent/SIS)*** expressed as percentage, and estimated by the ratio of the number of positive training events that strictly confirm the rule to the number of all positive examples. The number of examples that strictly confirm the rules is computed by applying the ***SIS*** (Strict Interpretation Schema).

Negative support (or ***negative coverage***) defined as the probability ***p(Premise | ~Consequent/SIS)***

Flexible support(T) (or ***flexible positive coverage***), defined as the probability ***p(Premise | Consequent/BFIS)***, expressed as a percentage, and estimated by the ratio of the number of positive training events that match the rule with a degree of match equal or greater than *T* to the total number of positive examples. The degree of match is computed by applying ***BFIS*** (Basic Flexible Interpretation Schema).

Confidence, defined as ***p*(*Consequent* | *Premise* & *SIS*)**--the probability, expressed as a percentage, of ***Consequent*** if ***Premise*** is strictly satisfied by an event, i.e., when the rule is evaluated by the ***SIS*** Interpretation schema.

Flexible confidence, defined as the probability, ***p*(*Consequent* | *Premise* & *BFIS*)**, expressed as a percentage, of ***Consequent*** if ***Premise*** is flexibly satisfied by an event, i.e., when the rule is evaluated by ***BFIS*** (Basic Flexible Interpretation Schema). This probability is estimated by using a training set of examples.

Training Confidence, defined as the probability ***p*(*Consequent* | *Premise* & *SIS, TS*)**, expressed as a percentage, of ***Consequent*** if ***Premise*** is satisfied by an event from the training set *TS* using the strict interpretation schema (SIS).

Note that if *p* and *n* are numbers of positive and negative examples strictly covered, and *P* and *N* are total numbers of positive and negative examples in the training set, then:

Support, briefly, ***Supp***, is p/P

Negative support, briefly, ***NegSup***, is n/N

Confidence, briefly, ***Conf***, is $p/(p+n)$

In addition to the above parameters, other parameters and rule characteristics can be included in the rule Annotation, depending on the specific implementation of the natural induction program. It can also be useful to annotate individual selectors in the rule, for example, by such parameters as *p* and *n*, which express the strict positive and negative coverage of the selector.

To give an example of an annotated attributional rule, and to illustrate the expressive power of $\mathcal{AC}$, let us consider a concept description expressed as a natural language decision rule:

If x_1 is less than or equal to x_2, x_3 differs from x_4, and x_3 is red or blue, then concept C. (42)

Using mathematical operators, rule (42) can be re-represented as:

If $x_1 \leq x_2$, $x_3 \neq x_4$, and x_3 is red or blue, then concept C. (43)

Let us suppose that attributes x_i, i=1,2,3,4 are all binary. To represent rule (43) as a decision tree, one would need a tree with 26 nodes and 20 leaves. To represent this rule using conventional decision rules (that employ only basic conditions), one would need 12 rules.

Suppose now that attributes x_i are five-valued. In this case, representing (43) would require a decision tree with 810 leaves and 190 nodes, or 600 conventional decision rules.

The need for such a complex representation of a relatively simple relationship demonstrates a major limitation of these representational formalisms.

Using attributional calculus, rule (43) can be represented as:

$$x_1 \leq x_2 \ \& \ x_3 \neq x_4 \ \& \ x_3 = \text{red v blue} \Rightarrow \text{Concept} = C \qquad (44)$$

which closely resembles (43), and could be easily translated to natural language rule (42).

Obviously, this example was specially constructed to illustrate $\mathcal{AC}$'s advantages, but it does point out some important properties of attributional rules. In a computer implementation of $\mathcal{AC}$ as a representation language for natural induction, a rule annotation can include more information than the basic parameters listed above, and there can also be similar annotation of individual selectors. In a system for natural induction it may be also advantageous to output rules not in the formal syntax of annotated calculus, but in a more natural form, easy to interpret and understand. For example, such an output might be in the form of a table:

[Concept is C] if:

	Supp	**NegSup**	**FSup**	**Conf**	**FConf**	**TCon**	**Pos**	**Neg**
$\mathbf{x_1 \leq x_2}$,, and	60%	7%	66%	56%	59%	61%	1200	350
$\mathbf{x_3 \neq x_4}$, and	56%	11%	60%	46%	49%	50%	1120	550
$\mathbf{x_3}$**= red or blue**	58 %	5%	60%	58%	51%	55%	1160	250
RULE	54%	3%	68%	87%	100%	100%	1080	150

***Additional annotation*:**
Total number of positive examples: P = 2000
Total number of negative examples: N = 5000
The number of uniquely covered examples: u = 980
Rule quality: Q = .9
Ambiguous examples: (pointers to examples covered also by rules in other rulesets)
Exceptions: (pointers to examples of other classes that satisfy the rule)
Positives covered: (pointers to positive examples strictly covered by the rule)
Flexibly covered: (pointers to all positive flexibly covered examples)

where:

Supp stands for Support
NegSup stands for Negative Support
FSup stands for Flexible Support
Conf stands for Confidence
FCon stands for Flexible Confidence
TCon stands for Confidence on the training set
Pos is the number of positive examples covered strictly by the rule
Neg is the number of negative examples covered strictly by the rule

If P and N are the total numbers of positive and negative examples in the training set, respectively then:

Supp = pos / P
NegSup = neg / N
Conf = pos / (pos + neg)

In the table above, annotations are assigned not only to the whole rule, but also to individual selectors. The table presents an easy to interpret representation of an attributional rule with annotations assigned to it. Such a rule representation has been used (approximately) in VINLEN, a multistrategy system for learning, data mining, and decision support being developed in the GMU Machine Learning and Inference Laboratory (Kaufman and Michalski, 2003).

11 SUMMARY

Attributional calculus is a logic and a representation system that combines aspects of propositional logic (by applying propositional operators to selectors), predicate logic (through the count and compound attributes), and multiple-valued logic (though flexible interpretation and multiple-valued logic operators). $\mathcal{AC}$ is specifically intended to support the development of natural induction systems that strive to hypothesize human-oriented descriptions from data.

$\mathcal{AC}$ provides a formal representation for inductive learning of descriptions that:

1. are compact, and easy to understand and interpret
2. can be simply and directly translated to natural language
3. are gradually generalizable through the internal logic operators and structured attributes
4. include compound attributes, a new type of attribute that permits the system to associate attributes with a specific object
5. include count attributes that permit the system to express simply a large number of different quantified statements
6. can be relatively efficiently implemented in a computer program.

The above features make Attributional Calculus particularly attractive for such areas as machine learning, data mining and knowledge discovery, knowledge mining, and inductive databases.

REFERENCES

Bloedorn E. and Michalski, R.S., “Data-Driven Constructive Induction,” *IEEE Intelligent Systems, Special Issue on Feature Transformation and Subset Selection*, pp. 30-37, March/April, 1998.

Bongard, M.M., *Pattern Recognition*, the original Russian edition appeared in 1967; English translation appeared in 1970, Spartan Books, New York, 1970.

Carney, J.D., *Introduction to Symbolic Logic*, Prentice Hall, Inc., 1970.

Clark, P. and Niblett, T., “The CN2 Induction Algorithm,” *Machine Learning* 3, pp. 261-283, 1989.

Cohen, W., “Fast Effective Rule Induction,” *Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, CA, 1995.

Copi, I.M., *Introduction to Logic*, Macmillan Publishing Co., Inc, New York, 1982.

Craven, M.W. and Shavlik, J.W., “Extracting Tree-Structured Representations of Trained Networks.” *Advances in Neural Information Processing Systems*, Denver, CO, MIT Press, pp. 24-30, 1996.

Craven, M.W. and Shavlik**,** J.W. **“**Rule Extraction: Where Do We Go from Here?” *Working Paper 99-1*, Machine Learning Research Group, Department of Computer Science, University of Wisconsin, Madison, 1999.

Duda, R. and Hart, P., *Pattern Classification*. New York: John Wiley and Sons, 1973.

Glowinski, C. and Michalski R.S., “Discovering Multi-head Attributional Rules in Large Databases,” *Tenth International Symposium on Intelligent Information Systems*, Zakopane, Poland, June, 2001.

Kaufman, K. and Michalski, R.S., “The Development of the Inductive Database System VINLEN: A Review of Current Research,” *International Intelligent Information Processing and Web Mining Conference*, Zakopane, Poland, 2003.

Kozy, J., Jr., *Understanding Natural Deduction: A Formalist Approach to Introductory Logic*, Dickenson Publishing Company, Inc., 1974.

Lavrac, N., and Dzeroski, S., *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, Ltd., 1994.

Liu, H. and Motoda, H., *Feature Extraction, Construction and Selection: A Data Mining Perspective,* Kluwer Academic Publishers, 1998.

Lloyd, J.W., *Logic for Learning: Learning Comprehensible Theories from Structured Data*, Springer, Cognitive Technologies Series, July 2003.

Medin, D. L., Wattenmaker, W.D. and Michalski R.S., "Constraints and Preferences in Inductive Learning: An Experimental Study Comparing Human and Machine Performance," *Cognitive Science*, 1988.

Michalski R.S., "Recognition of Total or Partial Symmetry in a Completely or Incompletely Specified Switching Function," *Proceedings of the IV Congress of the International Federation on Automatic Control (IFAC) (Finite Automata and Switching Systems)*, Vol. 27, Warsaw, pp. 109-129, 1969.

Michalski R.S., "A Variable-Valued Logic System as Applied to Picture Description and Recognition," in Nake, F. and Rosenfeld, A. (eds.), *Graphic Languages*, North-Holland Publishing Co. 1972.

Michalski R.S., "Variable-Valued Logic: System VL1," *Proceedings of the 1974 International Symposium on Multiple-Valued Logic*, West Virginia University, Morgantown, WV, pp. 323-346, 1974.

Michalski, R.S., "A Planar Geometrical Model for Representing Multi-Dimensional Discrete Spaces and Multiple-Valued Logic Functions," Report No. 897, Department of Computer Science, University of Illinois, Urbana, IL, January, 1978.

Michalski R.S., "Pattern Recognition as Knowledge-Guided Computer Induction," Report No. 927, Department of Computer Science, University of Illinois, Urbana, IL, May 1978.

Michalski R.S., "A Theory and Methodology of Inductive Learning," in Michalski, R.S., Mitchell, T. and Carbonell, J. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Co., Palo Alto, pp. 83-134, 1983.

Michalski, R.S., Concept Learning and Natural Induction, *Lecture Notes on Machine Learning and Applications*, Advanced Course on Artificial Intelligence, Chania, Greece, July 5-16, 1999.

Michalski R.S. and Kaufman K., "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, 2001.

Michalski R.S. and McCormick, B. H, "Interval Generalization of Switching Theory," *Report No. 442*, Department of Computer Science, University of Illinois, Urbana, 1971.

Michalski R.S. and Winston, P. H., "Variable Precision Logic," *Artificial Intelligence Journal* 29, Elsevier Science Publishers B.V. (North-Holland), pp. 121-146, 1986.

Pawlak, Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, 1991.

Pazzani, M., Mani, S. and Shankle, W.R., “Comprehensible Knowledge Discovery in Databases,” in Shafto, M.G. and Langley, P. (eds.), *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum, pp. 596-601, 1997.

Rine, D. (ed.), *Computer Science and Multiple-valued Logic: Theory and Applications*, North Holland, 1984.

Scholkopf, B. and Smola, A.J., *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.

Sebag, M., “Constructive Induction: A Version Space-based Approach,” *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 99), Vol. 2*, Stockholm, pp. 708-713, 1999.

Whorf, B., *Language, Thought, and Reality*, MIT Press, Cambridge, Massachusetts, 1956.

Wnek, J., “DIAV 2.0 User Manual: Specification and Guide through the Diagrammatic Visualization System,” *Reports of the Machine Learning and Inference Laboratory*, MLI 95-5, George Mason University, Fairfax, VA, 1995.

Wnek, J. and Michalski, R.S., “Experimental Comparison of Symbolic and Subsymbolic Learning,” *HEURISTICS, The Journal of Knowledge Engineering*, Special Issue on Knowledge Acquisition and Machine Learning, Vol. 5, No. 4, pp. 1-21, 1992.

APPENDIX

Effect of the Representation Language on the Choice of Hypothesis in Inductive Learning

This Appendix provides a simple illustration of the importance of representation language in inductive inference. Suppose that an inductive system is learning concept *z*, and has to choose between two competing hypotheses, **A** and **B**, which cover the same training examples, denoted by "+" and "-", as shown in the diagram in Figure A1. The small digits in the lower-right corner of each cell denote the number of positive literals in the expression corresponding to the cell. Hypotheses **A** and **B** are illustrated in Figures A2 and A3, respectively.

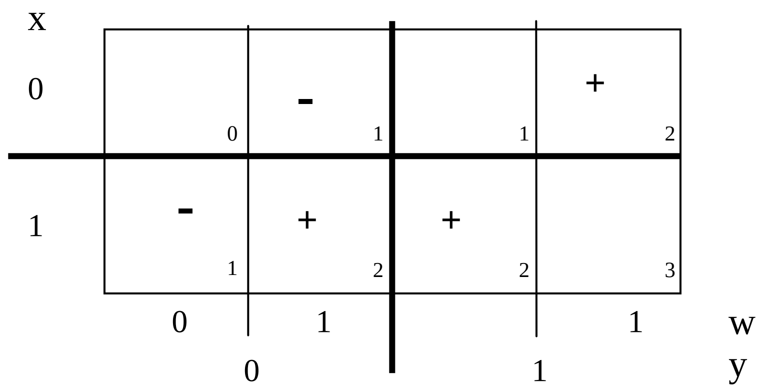


Figure A1: Training examples.

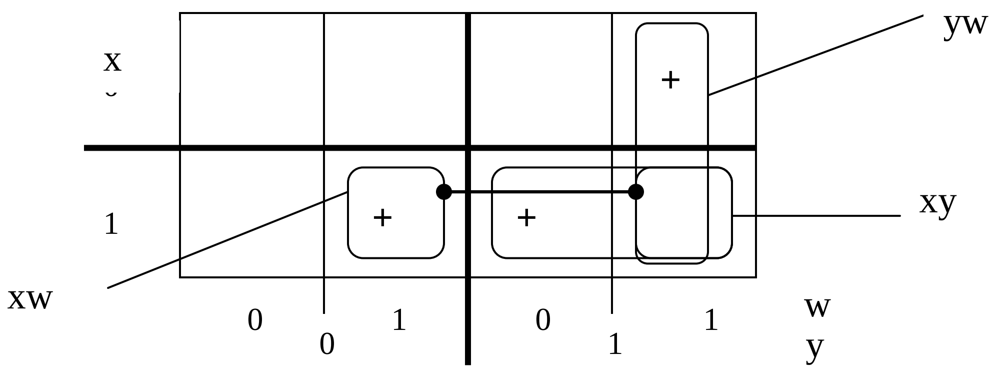


Figure A2: Hypothesis A.

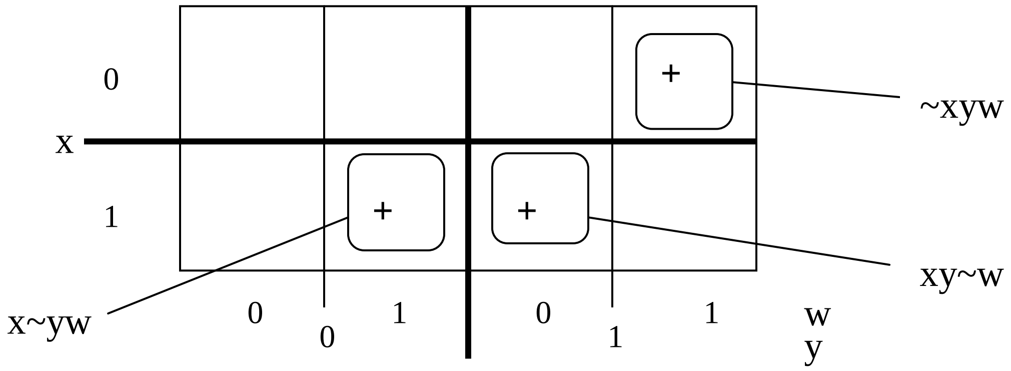


Figure A3: Hypothesis B.

Hypotheses **A** and **B**, if expressed in propositional logic, are:

$$\mathbf{A}:\ z \Leftarrow x\sim yw \vee xw \vee \sim xyw \qquad (A1)$$

$$\mathbf{B}:\ z \Leftarrow x\sim yw \vee xy\sim w \vee \sim xyw \qquad (A2)$$

While expressions (A1) and (A2) are legal both in propositional and attributional calculus, in the latter language they can be alternatively represented using a count attribute:

$$\mathbf{A}:\ z \Leftarrow [\mathrm{count}(x,y,w) \geq 2] \qquad (A3)$$

$$\mathbf{B}:\ z \Leftarrow [\mathrm{count}(x,y,w) = 2] \qquad (A4)$$

These expressions can be validated by observing the small numbers in the corners of the cells in Figure A1. Expression (A3) states that if there are two or more attributes in {x, y, w} that take value "1" in an example, then the example is assigned class *z*. Expression (A4) states that if there are exactly two attributes in {x, y, w} with value "1," then the example is assigned class *z*.

Hypotheses **A** and **B** can also be expressed using logically equivalent decision trees.

```
A:   If y=0, then
         if x=0 then ~z, else
             if w=0 then ~z
             else z
     else:
         if x=1 then z, else
             if w=1, then z
             else ~z                                  (A5)
```

```
B:   If y=0, then
         if x=0 then ~z, else
             if w=0 then ~z
             else z
     else:
         if x=1 then
             if w=0 then z,
             else ~z
         else:
             if w=1, then z
             else ~z                                  (A6)
```

Note that the decision trees (A5) and (A6) appear to be significantly more difficult to understand and represent mentally than corresponding attributional calculus expressions (A3) and (A4).

The decision trees (A5) and (A6) are represented graphically in Figures A4 and A5, respectively. In these trees, the left branch stemming from each node is assigned value 0, and the right branch is assigned value 1 of the attribute assigned to the node.

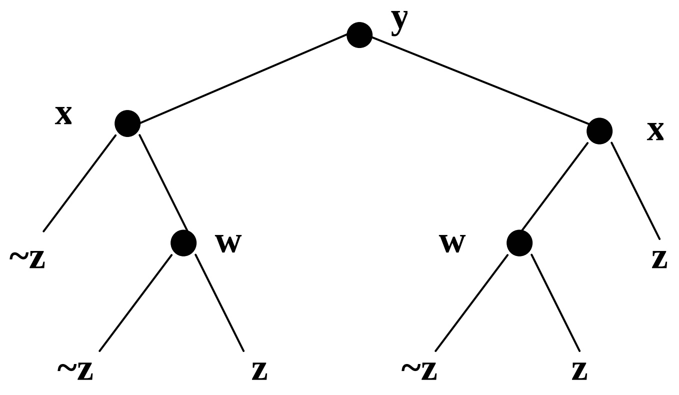


Figure A4: A decision tree representing hypothesis A.

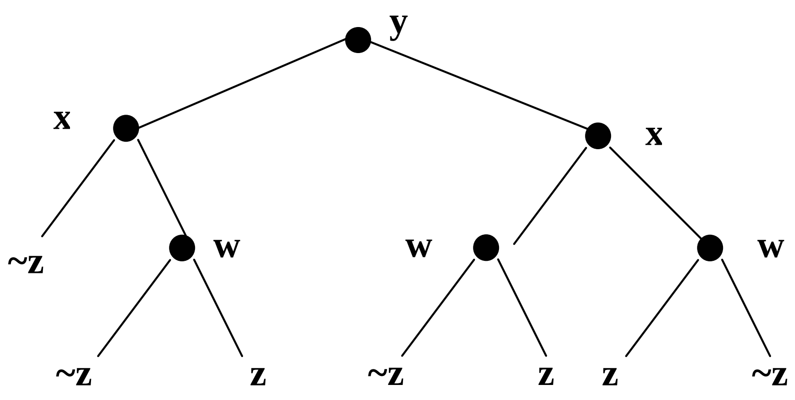


Figure A5: A decision tree representing hypothesis B.

As one can see, the decision tree in Figure A5 is somewhat more complex that decision tree in Figure A4. The one in Figure A4 has five nodes and six leaves, and the one in Figure A5 has six nodes and seven leaves.

Expressions (A1), (A3) and (A5) representing hypothesis **A** are all logically equivalent, as are expressions (A2), (A4), and (A6) representing hypothesis **B**. Because they all are complete and consistent with regard to the training data), the question arises as to which hypothesis from each pair {A1, A2}, {A3, A4}, and {A5, A6} would be selected by learning programs using these description languages. When two hypotheses are both consistent and complete with regard to the training data, an extra-logical criterion needs to be applied. In inductive learning, a typical such criterion is hypothesis simplicity, often called "Occam's razor": If two hypotheses explain the data, choose the simpler one.

It should not be controversial to assume that the hypothesis (A1) is simpler than (A2), because it has fewer literals; that hypothesis (A3) is more complex than (A4), because a relational operator is more complex than the equality operator; and that (A5) is simpler than (A6), because it has fewer nodes and leaves. Thus, programs using propositional calculus and decision tree representations would select hypothesis **A**, and a program using attributional calculus might select hypothesis **B**. Choosing hypothesis **A** or **B** thus depends on the representation language used by the learning program. This example thus demonstrates that using different hypothesis representation languages can lead to generating different hypotheses from the same training data.

A publication of the *Machine Learning and Inference Laboratory*
School of Computational Sciences
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: K. A. Kaufman

The *Machine Learning and Inference (MLI) Laboratory Reports* are an official publication of the Machine Learning and Inference Laboratory, which has been published continuously since 1971 by R.S. Michalski's research group (until 1987, while the group was at the University of Illinois, they were called ISG (Intelligent Systems Group) Reports, or were part of the Department of Computer Science Reports).