ANALYSIS++: Object-Oriented Framework for Multi-Strategy Machine Learning Methods

Edward S. Blurock
Johannes Kepler University,A-4040 LINZ, Austria
email: blurock@risc.uni-linz.ac.at

July 14, 1998

Abstract

ANALYSIS++ is a framework, written in **C++**, for the manipulation and development of multi-strategy machine learning methods and their corresponding data objects. The framework consists of a heirarchy of subframeworks, an object framework for the management of the data used and generated by machine learning methods and their further use by other methods, an algorithm framework for the organisation of the machine learning methods and their support routines and a goal framework for the management of algorithms in a goal-oriented strategy. [1]

1 Introduction

Data analysis involves extracting general principles from a history of known samples. Machine learning is a special class of these algorithms with two distinguishing features:

More Human Approach Machine learning techniques usually try to mimic humans either in a biological way, such as neural nets, or in a psychological way, such as those based on logic and reasoning.

Optimisation Machine learning algorithms usually rely on optimisation or search, in contrast to, for example, the deterministic matrix operations of multivariate statistical.

In practice, in all areas of data analysis, only one algorithm is chosen which is particularly general or particularly suited for the problem at hand. Getting, for example, a final prediction function, involves optimising the results within this algorithm. However, due to the fact that 'miracle algorithms' don't exist, a shift to a multi-strategy approach, to combine different types of analyses, has been taken. More importantly, it has been recognized that each data analysis algorithm can potentially contribute new information (also using many results together can improve accuracy). There are many classes of data analysis techniques and each one is based on a different philosophy[7, 2, 5]. Each views the data differently and recognises different patterns within the data. One example is clustering objects into classes of similarity. The traditional clustering algorithm[8], using a distance function as a measure of similarity, can yield different results with different distance functions, each with a different definition of similarity. In addition, the differentiating between agglomerate, that is starting with the individual objects and then pairing them up into clusters, or divisive algorithms, starting with all the objects and breaking them down into clusters, yields different results, or incremental machine learning methods, placing object individually into clusters as they are introduced[3, 4], yield different clusters because they all differ in approach and in their definition of similarity. Hence they all yield different insight as to how and which objects are similar.

[1] sponsored by: ESPRIT Project 22897, UNI-SOFTWARE PLUS, A-4232 Hagenberg, Austria

2 Machine Learning Analysis

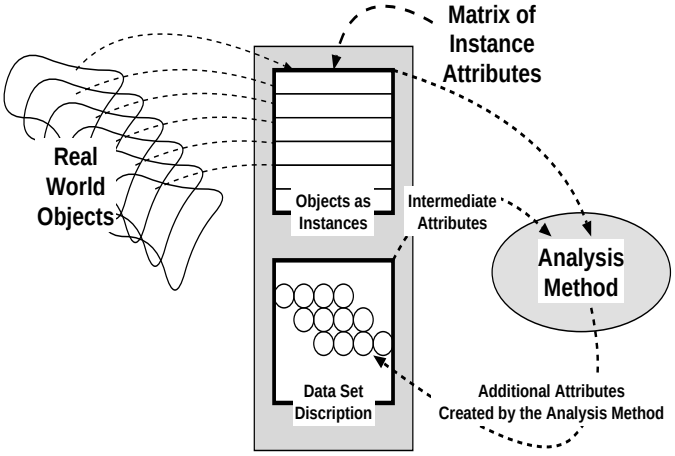


Figure 1: The learning cycle translating the 'real world' into a dataset and using analysis methods to find 'descriptions'.

Figure 1 shows one learning cycle starting with the set of 'real' objects analysing their translation in the dataset to find new descriptions, such as a predictive function or classification. The complete 'learning' process consists of many such cycles, each cycle creating a new set of descriptions. In the multistrategy approach[6], each cycle can involve a different analysis method and the descriptions are reused. At the center of this cycle is a data set with two distinct parts:

Instances The history of samples to analyse are the dataset instances. Each instance corresponds to a real object. The attributes within the instances are a representation of the objects, i.e. an approximation of reality. This set of instance attributes is used by the machine learning methods to compare and distinguish between the dataset instances to gain insight into the system.

Descriptions The result of machine learning algorithms is a set of descriptions of the data as a whole. Typically, in a learning cycle, a set of descriptions are suggested (maybe by a previous analysis method) and it is the job of the learning method to pick out and combine these descriptions into a generalisation of the whole data set.

In the multi-strategy approach, a description generated by one method can be used in the next method. After the final analysis it contains a characterisation of the system and the predictive functions representing the generalities and intuition gained in each learning cycle. The data set description have two basic purposes:

Intuition They provide further information about the system. If they are in the form of readable rules, these can be interpreted to enhance intuition about the system. For example, those parameters or descriptions used in a characterisation from an analysis method could be considered significant and those not used could be considered ignorable. Another example would be that a given condition holds only when $X > 45.3$, which could be a quantitative justification to the original intuition that 'higher' values of X were needed[1].

Intermediates Each analysis method requires distinct data structures as input which often differ from the 'raw' information available. For example, most rule (logical description) induction methods require a list of possible predicates as input. A transformation from, for example, a set of real numbers (the values of a parameter, X) to a set of predicates, $X > 2.0$, $X < 5.3$, $6 < X \leq 11.2$, must be

made (Additional preprocessing analysis can also determine suitable parameters to use and suitable boundaries).

Each analysis method can deliver a different data structure which, in turn, can give additional information or justification of previous results. The goal of multistrategy learning is to be able to use and exchange the methods and their corresponding data structures efficiently. An essential aspect of the multistrategy approach is to examine a method or data structure to see what information can be extracted from it.

For example, a clustering method builds a hierarchy of 'similar' objects and as one moves down this hierarchy the objects become more similar. Within this data structure, many types of information can be extracted. For example, the original dataset may be too large (in number) or too diverse (in conditions and situations) and each cluster could be used as a smaller more consistent set for further analysis. The automatic isolation of parameter values to be used as boundaries for the predicates (such as $X < 3.54$) could be automatically accomplished by one-dimensional clustering using the bounds of each cluster as the bounds used in the predicate. An examination of the descriptions of the clusters themselves could reveal the 'significant' parameters or descriptions. If a cluster is formed where 85% of the objects have $X > 85.9$ (as opposed to a random distribution in the parent cluster), then the significance of the parameter `X` is established and if the description, $Y > 3$, has the same distribution as the parent then a hint is given that the parameter `Y` is less significant. Furthermore, under the principle that similar object behave similarly, following an object down a hierarchy of clusters could give a prediction on about its behavior (This is a generalisation of Case-Based reasoning).

2.1 Learning Within the Framework

There are two important criteria in developing a framework supporting the multi-strategy approach:

Reuse of Object Information In most cases, the information within an object created by an analysis technique can be used as input to other methods. The framework should support this by providing a means of communication between methods, sharing of results and use of results in different contexts.

Organisation of Tasks A complex algorithm may consist of many subalgorithms or tasks. When implementing an algorithm, one should be able to concentrate on the algorithm itself. Furthermore, the exchange of algorithms within a complex task and the reuse of algorithms in different contexts should facilitated. This also facilitates experimentation of new combinations of methods.

One of the aims of the current work is to provide an efficient experimentation framework for the use of algorithms and their combination and the objects they produce. The complete ANALYSIS++ framework is made up of a hierarchy of the following subframeworks:

Objects The object framework forms the basis of objects available for attribute (and result) descriptions of the instances under analysis. A hierarchy of objects is provided to promote the exchange of objects between algorithms.

Algorithm The algorithm framework represents the characterisation and modularisation of the tasks within machine learning algorithms and the supplementary tasks needed for their setup and use.

Run The run framework manages the execution of algorithms and the data objects produced. Modularisation of tasks is promoted through the separation of task prerequisites and the actual job that the task has to perform.

3 Object Framework

Part of the generalisation and development of new algorithms is the definition of more general or new data types. For example, algorithm is defined with respect to a basis class, but can be applied to any class derived from this class. This is done by defining framework classes (defined at compile time) which can be combined and instantiated to new data types (defined at runtime).

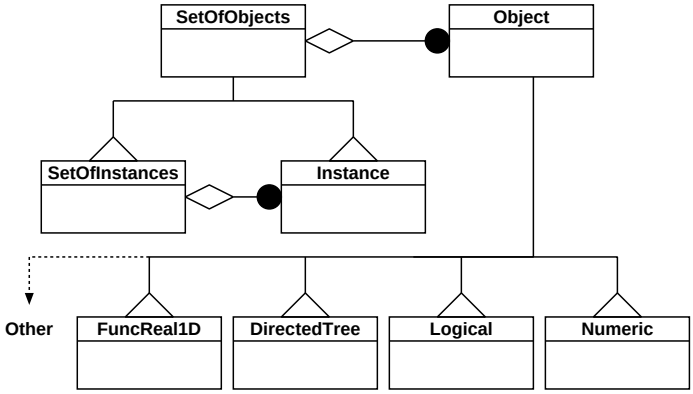


Figure 2: All objects stem from the basic object type. The data has two levels, one for objects that are valid for the entire data set and one for the set of objects for example example 'instance'.

The representation of the instances and descriptions is shown in the class structure of Figure 2. The classes of the machine learning framework are built using a set of basis classes based on the STL-template library. The base class of the framework is the `Object` class. The `SetOfObjects` class holds lists of classes derived from `Object`. `Instance` is a `SetOfObjects` with the attributes descriptions of the instance. The `SetOfInstances` class holds the matrix of instances and attributes (i.e. the set of instances) and is itself a `SetOfObjects` holding the descriptions over the entire data set. `SetOfInstances` corresponds to the data of Figure 1.

3.1 Expandable Class Structure

Part of new algorithm development work is the expansion and further organisation of the basic object types and how these expanded data types can be used within the available machine learning algorithms. This is done by defining framework classes with can be instantiated to new types.

Associated with each attribute is an attribute type describing its fundamental characteristics. The foundational classes provide the 'hard-coded' classes from which attribute types can be derived. Some attribute types are the foundational classes themselves and some are defined by the user (at runtime). The basic relationship between the foundational classes and the attribute types is as follows:

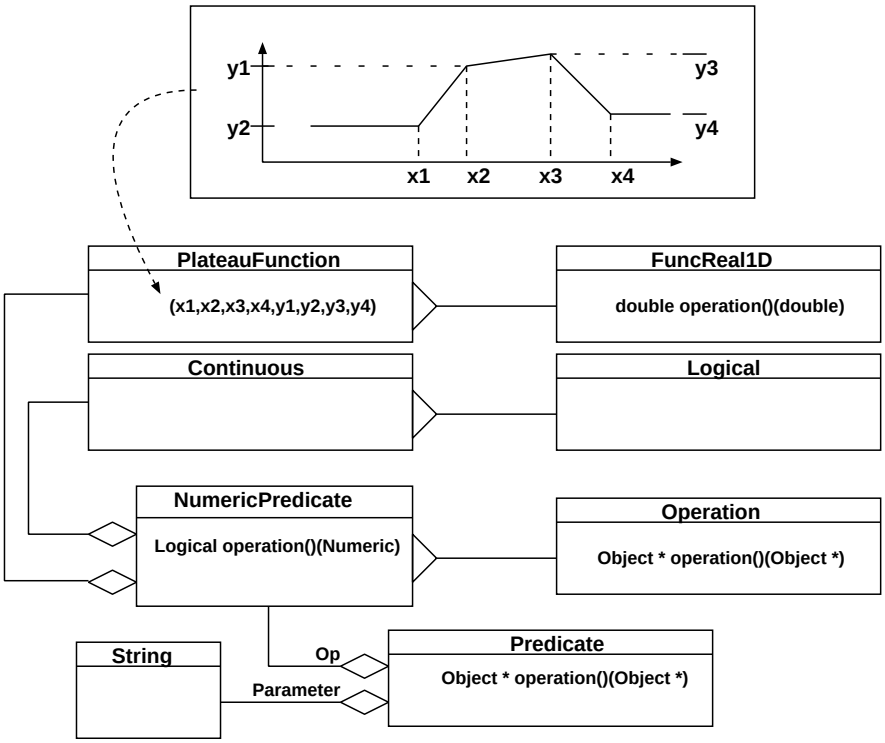
Foundation Class This is the hard coded (compiled) class available. The new classes are made of combinations of these classes.

Attribute Type When the foundational class is instantiated, a specific attribute data type is defined. In some foundation classes some information is left open to be specified at runtime. In this way new types can be defined.

Attribute Type Instance This is the value belonging to the corresponding attribute data type that actually would be assigned to an specific attribute in an instance.

An simple example is a nominal attribute representing a set of colors. The foundational class for any nominal attribute is the class `NValuedLogic`. The instantiation of this class to the attribute type of colors is the three-valued logic class instantiated with the three labels, `red`, `green` and `blue`. A specific attribute value (as used in an dataset instance) of this type could be, for example, `green`.

Some foundational classes are compound classes consisting of several other foundational classes. The instantiation to a specific attribute type would be the instantiation of the classes making up the compound class. For example, within Figure 3 the foundational class `NumericPredicate` is a predicate function in which a real value is mapped to a logical value. It main use is to form fuzzy logic predicates. Its definition needs a logic class (`Logical`) and a numeric function class (`FuncRead1D`). The logic class specifies the form of




```
%% Definition of Attribute Types:
%% 'FuzzyPredicate' derived from 'NumericPredicate'
%%            composed of 'Continuous' logic and a 'PlateauFunction'
DataType:  NumericPredicate A sample predicate
           FuzzyPredicate           Continuous       PlateauFunction
%% 'FuzzyParameterPredicate derived from 'Predicate'
%%            composed of 'FuzzyPredicate' (as previously defined)
DataType:  Predicate        A predicate of a numeric predicate function
           FuzzyParameterPredicate   FuzzyPredicate
%% Instantiation of FuzzyParameterPredicate: The 'price' is around 10000 dollars
Predicate:
price
-Infinity T +Infinity T PlateauFunction 0.0 1.0 1.0 0.0  8000 9000 11000 12000
```

Figure 3: The interaction and building of a complex data structure from the frame given is shown. The 'Predicate' object has two strings for the parameter name (to be found for each example instance) and the logic type of the predicate. The predicate itself is made up of a numeric predicate (a function mapping a number to a logic value). The numeric predicate, in turn, has two slots for the logic type and the numeric function type.

the logic that the predicate will return and the numeric function specifies the functional dependence, which, for fuzzy logic, maps the real numbers to a number between zero and one. In the example, the logic class is set to `Continuous` logic, a fuzzy logic class[9] having a 'logic' value from zero to one and the `FuncRead1D` call is set to `PlateauFunction`, where 8 parameters determine the exact fuzzy relationship. This specifies a new class derived from `NumericPredicate`. This class could be used to define many fuzzy relationships, each instantiation of which would be a specific set of the eight parameters.

A common description used in machine learning asks a question about a specific parameter; for example, is the parameter `X` (of an instance in the dataset) more or less greater than 2 (a fuzzy logic predicate). The `Predicate` foundation class is used to define such questions and is composed of a logical `Operation`. In the instantiation of the data type, the name of the parameter `String` and the logical operation `Operation` must be specified. As shown in Figure 3, one such `Predicate` could be specified with a `NumericPredicate` (as explained earlier). For example, the predicate `X` (fuzzy) greater than 2 would be instantiated with the parameter string set to `X` and the 8 parameters in `PlateauFunction` set appropriately to the ramp function around 2.

3.2 Hierarchy of Data Structures

An essential part of the multistrategy approach is the reuse and exchange of information that is produced by a given machine learning method. What information is available and how it can be used later requires extensive analysis, before implementation, of the components and type of method and the data structures it produces.

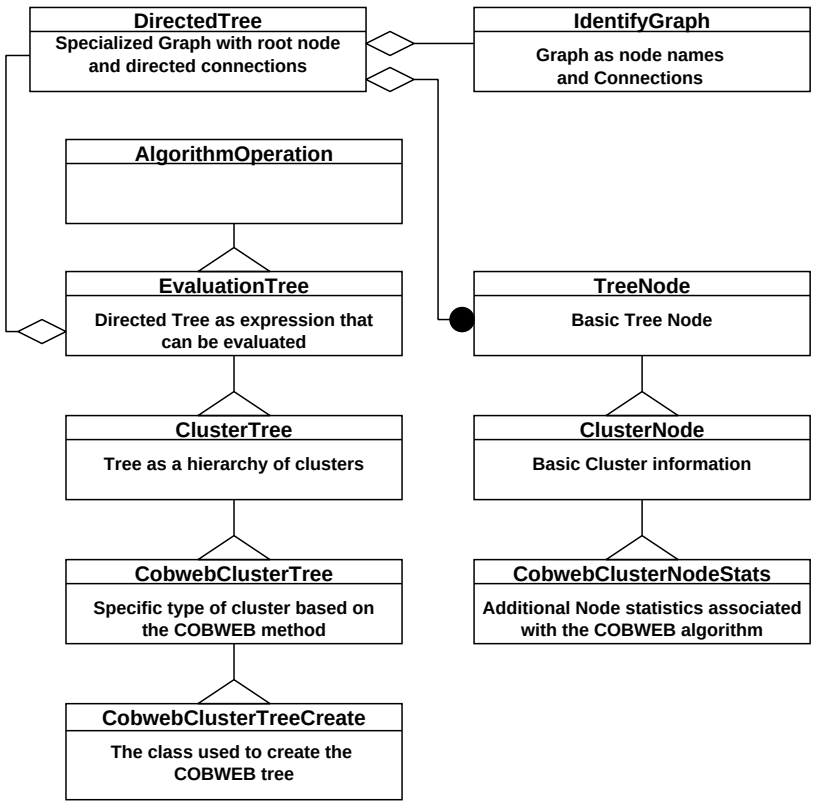


Figure 4: Part of the organization of the algorithms is the hierarchy of functionalities. The algorithm is interpreted at the different levels in different ways. Going down the hierarchy specifies addition functionality and interpretations.

On way to facilitate using the information in a data structure in many ways is to define the data structures

in a hierarchy of increasingly detailed classes. At each class level, a given aspect of the data structure is available. Progressing up the hierarchy, more detailed and hence other aspects of the data are made available. A simple example is shown in Figure 4. The resulting data structure is, at one level, an `EvaluationTree`. Here, the tree structure is used to make predictions about other 'new' objects and how they fit in the tree. At the next level, `ClusterTree`, the data object is considered a tree of clustered objects. Here, information about the clustering, for example, which objects are grouped together and at what level, can be accessed. The next level is the produced directly by the method itself, `CobwebClusterTree`. At this level a specific description of the individual clusters is available. The last level, `CobwebClusterTreeCreate` is the class which is used to create the tree from the dataset[4]. A similar hierarchy is shown for the nodes within the trees.

The natural structure of the relationships (and commonalities) between the different structures that different machine learning algorithms are mirrored. It is, however, left to the developer to find, through extensive analysis, the new method's place in the hierarchy. If a concept does not pass into the well defined hierarchy, then coercion would be used to create a data structure that is in the hierarchy. It is especially important in the implementation of the initial set of algorithms of a given type that an initial efficient and useful structure is given on which others can be built. It is a major goal of the framework to provide a means to experiment with implementation structures.

4 Algorithm Framework

The algorithm subframework is for the characterisation and modularisation of tasks. It structures the selection of the set of available 'hard-coded' operator and algorithm classes through characterisation within a selection tree mechanism. The philosophy is to isolate tasks and to provide a mechanism to form complex algorithm combinations. Within this framework distinctions will be made between three types of operators:

Algorithm This is a single complex operator accepting the data set of instances and the current execution status as input. A particular machine learning algorithm would come under this category. A single algorithm consists of a collection of subalgorithms and operations.

Subalgorithm A subalgorithm, though it looks exactly like an algorithm in its definition, is meant to be a smaller task within a more complex algorithm.

Operation An operation is an operator acting on one or two data objects resulting in another data object. This represents a simple operation within an algorithm or subalgorithm.

For example, the main algorithm could be a given cluster algorithm. First, this needs an operation, the distance function. This is an operation because the distance function defines the distance between two data objects and does not use the entire set of instances or attributes. The second requirement is the subalgorithm defining how the clustering should be accomplished. This is indeed a subalgorithm because it involves the entire set of instances and algorithms.

To promote modularisation and ease of implementation (and to reduce implementation errors) an algorithm is structured into six basic member functions, each of which has a well defined and isolated task:

SetUpAlgorithm This selects the subalgorithms to use from the algorithm selection tree (See section 4.1).

CheckInput This checks whether the required input parameters are available.

SetUpInput This instantiates the input parameters

Calculate This is the heart of the algorithm and assumes that all the necessary information has been set up.

WriteOutputValues This writes out the results as intermediate values (See section 5)

ConcludeRun This concludes the run by cleaning up objects used by the algorithm.

This breakdown structures (and isolates) the essential tasks associated with algorithm execution.

4.1 Algorithm Selection Tree

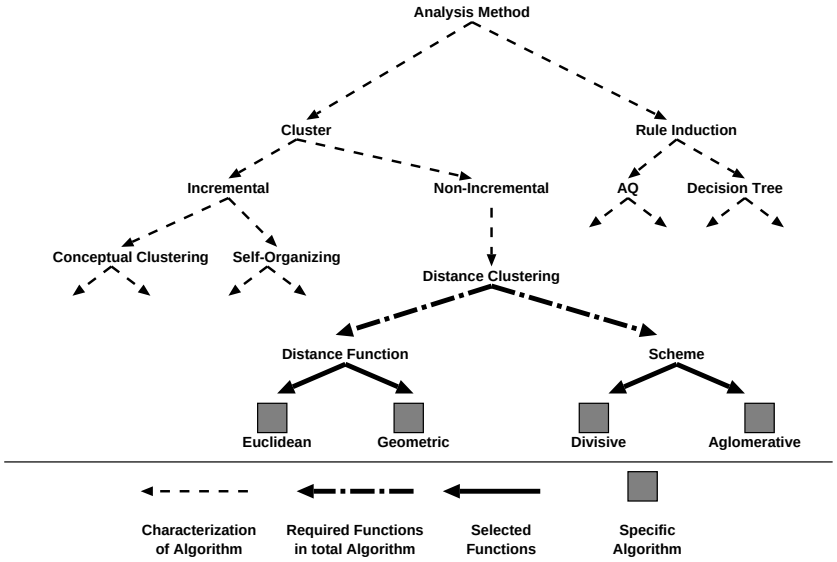


Figure 5: The tree structure provides an organisational level on top of the set of individual algorithms. The top level is the characterisation levels, which are keywords isolating the type and general function of the algorithm. At the next level, a particular algorithm may be composed of a set of subalgorithms which can be selected.

The goal of the algorithm selection framework is to provide an organisational structure to the classes of algorithms. At compile time, the entire set of algorithms is an unstructured list with no interrelationships or characterisations. The selection tree is built at runtime to dynamically structure the algorithms to suit the particular application. The organisation is a three level hierarchy:

Characterisation Through external analysis of the algorithm, the purpose, resulting data structure and type of the algorithm is determined. This information is structured in a hierarchy of keywords and gives a 'semantic' characterisation and organisation of the algorithm.

Selection Given a category or type, the selection of the specific algorithm begins. This level refers to a specific implemented algorithm.

Configuration A specific algorithm may be a composite of several subalgorithms; Which set is needed is given at the level directly below the 'selection' level. The configuration pointing to the specific algorithm can have many levels.

The key to the organisational structure of the algorithms is the 'algorithm selection tree'. Figure 5 shows a portion of a selection tree specifying a cluster algorithm. All the non-leaf nodes in the tree are keywords and can be specified by an ASCII or GUI interface. Depending on the position in the tree the keywords are either characterisations, selections or configurations. The leaves of the tree are the selected algorithms. The characterisation (as shown in the example) could refer to the type of algorithm, like 'clustering' or 'rule induction'. However, the purpose of the analyses could also be characterised, like 'attribute reduction', specifying methods that can be used to reduce the number of attributes, or 'dataset simplification', to cluster a disperse set of instances into more similar groups for more exact analysis; in future uses, these could be the basis of a more sophisticated use characterisation involving 'reasoning'. It should be noted that the leaf nodes of the selection tree are not limited to `Algorithm` classes and could be used as a general selection mechanism for any data structure.

The selection level refers to a specific algorithm. If the algorithm is made up of or needs several subalgorithms, then the configuration of the algorithms guides the choice. The nodes directly below the 'selection' algorithm are the categories of algorithms needed. Under these nodes is a hierarchy leading to specific algorithms. For example, within a given clustering algorithm, there are two choices of operations making up the algorithm, the distance function, for example, geometric or euclidian, and how this distance function should be used, for example, agglomerative or divisive.

5 Run Framework

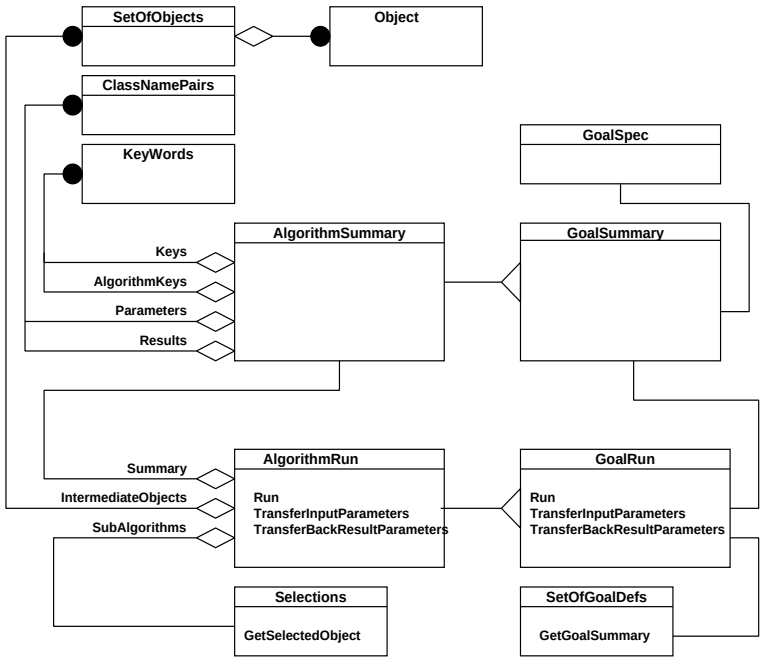


Figure 6: The object structure of the goal and algorithm framework is shown. The goal structures encapsulates algorithm information.

The `Run` framework manages the execution of algorithms and the access to data objects; Figure 6 shows the class structure. The `AlgorithmRun` framwork specifies and runs an algorithm and the `GoalRun` framework, inheiriting the `AlgorithmRun` framework, gives an added semantic to the algorithm by defining it as a goal; each of these has essentially two levels:

Summary The `AlgorithmSummary` and `GoalSummary` classes use string name specifications (`KeyWords` and `ClassNamePairs`) allowing for a straightforward interface specification (ASCII or GUI).

Run The `AlgorithmRun` and `GoalRun` frameworks hold the instantiated input, intermediate and output objects used during the execution of a goal or run.

5.1 Algorithm Framework

The `AlgorithmSummary` holds the essential information for a complete algorithm specification:

AlgorithmSelection This is the set of keywords used to traverse the algorithm selection tree to select the subalgorithms and operations used by the algorithm.

Keys To guide the execution of an algorithm, keywords can be used to make decisions. This is a list of string labels.

Parameters Some algorithms need extra parameters which, in principle, can be any data type. This specification gives a list of labels (and their corresponding type) to be accessed from the dataset descriptions when the algorithm is run.

Results This the list of data structures, created by the algorithm, that will be transfered back to the dataset.

This specification consists of keywords or lists of keywords. Instantiation of this specification, the `AlgorithmRun` class, comes from the dataset of instances and descriptions and the algorithm selection tree. All the intermediate results are stored within this class and the results of the algorithm come from this class; The essential member functions run the algorithm, `Run`, transfer the needed input parameters from the dataset, `TransferInputParameters`, and transfer back the resulting parameters, `TransferBackResultParameters`. The input to these routines reference the entire dataset.

5.2 Goal Framework

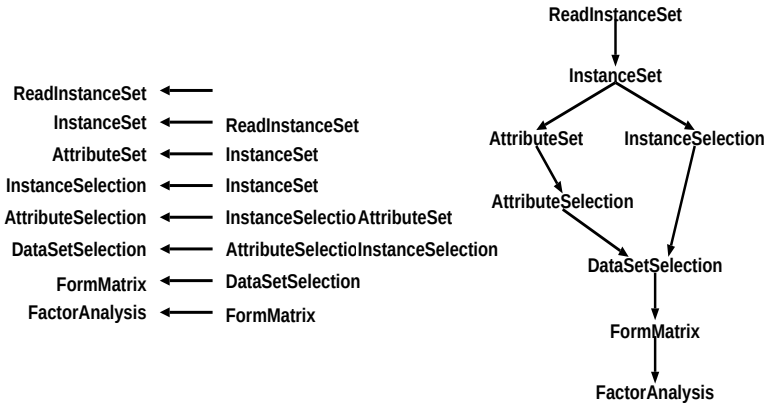


Figure 7: The set of goals as a set of rules and the execution sequence they represent is shown.

The goal framework inhierits the algorithm framework and coordinates the interaction between algorithms and gives a level of characterisation to the algorthm. Associated with each goal is an algorithm and a set of prerequisite goals. The prerequisites of a goal would be 'resolved' (several levels of resolution may take place because the immediate prerequisite goals themselves may have goals to satisfy) and satisfied (through execution of their corresponding algorithms) before the current goal algorithm is executed. At present the use of goals and goal satisfaction is an organisation tool managing algorithmic flow. Figure 7 shows an example of a set of goals with prerequisites leading to an execution order. The goal organisation is not defined at compile time, but rather is a database of goals determined at runtine. The following is a brief summary of the effects and uses of a goal oriented algorithm arrangement:

Decision Making Goals and their resolution provide a higher order means of decision making. It is a prerequisite for the use of 'reasoning' in deciding which tasks to perform.

Interface Simplification The definition of a higher order goal is an efficient way to specify a task.

Task Resolution Goals are an efficient way to express high order tasks that are to resolved in the details of actual execution. It provides a mechanism to shield the user from details.

Prerequisites This is the simpliest form of task resolution. Before a specific algorithm can be executed, usually specific data structures need to be present.

Task Bookkeeping The management of tasks to be done and those already done can be simplified. For example, in the execution of previous tasks, the prerequisites for the current task could already have been accomplished.

Task Definition In its simpliest form, a goal can be used as a higher level command representing a common task.

Distributed Programming Due to the modular nature of the goals, their structuring in a distributed environment, i.e. tasks being dealt with by other processes, is facilitated.

5.3 Flow of Execution

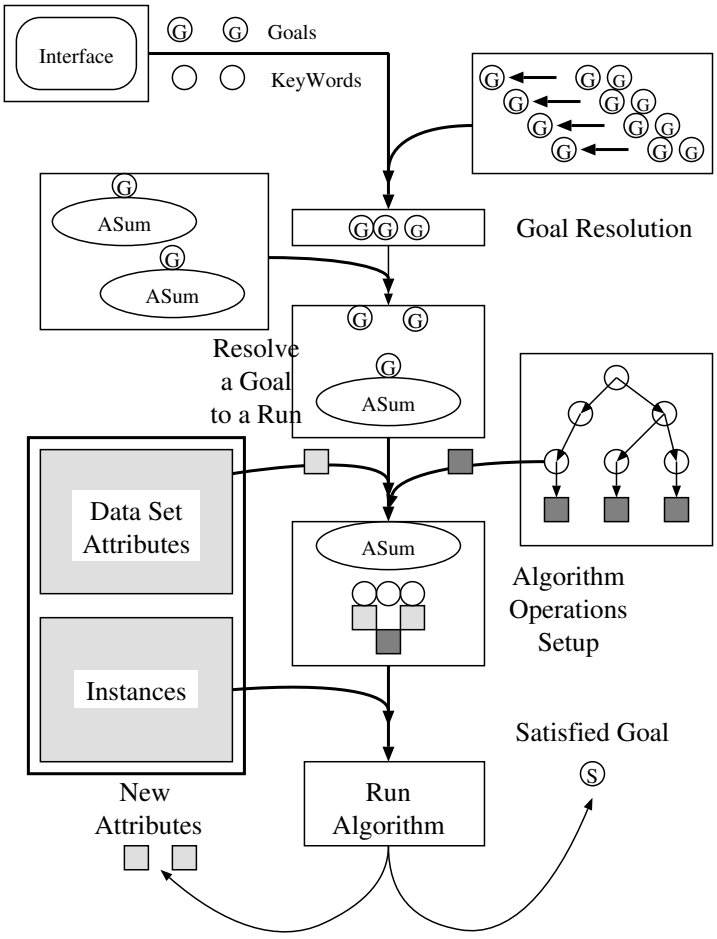


Figure 8: The flow of interface, goal resolution, algorithm selection, algorithm execution and thereby generation of new objects is shown.

Figure 8 shows the flow of the execution of a single task. The interface builds up the set of goals to be executed. These consist of goal keywords (and can be easily formulated as interface buttons or ASCII input). The goals are resolved to determine the prerequisite goals needed before execution of the main goal can begin. The procedure for the execution of these subgoals proceeds in the same way. From the goal keyword, the goal definition, which included the definition of the associated algorithm, is found. From the `AlgorithmSummary` information, the necessary algorithm and subalgorithms are extracted from the `SelectionTree`. Furthermore, the set of input parameters needed are collected from the dataset. At this point the algorithm can be run. The main input to the algorithm `Run` routine is the set of available instances. At the end of the run, the goal is satisfied and the resulting parameters are given back to the dataset.

6 Discussion and Future Directions

The ANALYSIS++ system is a framework supporting the multistrategy approach. An important aspect of this support is the modularisation and configuration of tasks. Each individual task is kept small enough to be manageable. Furthermore, breaking down large tasks into smaller tasks promotes flexibility in combining algorithms in different ways as needed. All configurations, whether it be structuring the set of algorithms, configuring the algorithms and goal relationships or executing a specific analysis problem, are done using keywords which can come from either an ASCII or GUI interface.

The division of the `Object`, `Operation`, `Algorithm` and `Goal` components allows, in an implementation, the developer to concentrate on a single task and other details are transparent or assumed. For example, in the development of a new machine learning algorithm, details of the dataset setup, parameter availability, prerequisite operations, how it is called and other tasks not directly associated with the specific algorithm can be assumed and the developer can concentrate on the details of the new algorithm. Another aspect of the modularisation is a (restricted) interface of function definitions. This includes not only the allowed functions at each level of a hierarchy of objects, but also, a more artificial, breaking down of a function definition into its component parts.

6.1 Future Work: History Framework

The history framework, through the automatic cataloguing and storing of the current analyses in a 'database', promotes the multistrategy approach by making their results are available for further use. The history framework will be a major emphasis in future expansion of the machine learning framework. The history would interface similarly to the goal and algorithm run frameworks. The essential difference would be that the lifetime of data objects would go beyond a single goal resolution. It would help in the management of reuse of objects, a task that is done manually (through the output of data structures after a run) now.

Within this framework, the analysis process of the dataset is viewed as a 'search', in the classic artificial intelligence sense, for the best possible set of analyses. This means that, within this search some sequences of analyses could prove to be fruitless, forcing another branch of analyses (backtracking). The history of analyses is basically a 'search tree', where each node is a new step in the analysis. Each node stores the results of the analyses and a characterisation of what analysis was done. It is also forseen that characterisation of the data objects and algorithms, for example through the selection tree or goal classes, could be used as a basis for 'reasoning' about which steps should be taken or which objects should be used.

References

[1] Edward S. Blurock. Automatic learning of chemical concepts: Research octane number and molecular substructures. *Computers Chem.*, 19:91–99, 1995.

[2] Garry Briscoe and Terry Caelli. *A Compendium of Machine Learning : Symbolic Machine Learning*, volume 1 of *Ablex Series in Artificial Intelligence*. Ablex Pub Corp., 1995.

[3] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[4] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. In J. G. Carbonell, editor, *Machine Learning: Pardigms and Methods*, pages 11–61. MIT Press, 1990.

[5] Pat Langley. *Elements of Machine Learning*. Morgan Kaufman, 1995.

[6] Ryszard Michalski and Gheorghe Tecuci. *Machine Learning: A Multistrategy Approach*, volume 4. Morgan Kaufmann, 1994.

[7] Ryszard S. Michalski and Robert E. Stepp. *Machine Learning: An Artificial Intelligence Approach*, chapter 11. Morgan Kaufman, 1983.

[8] J.T. Tou and R.C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, 1974.

[9] Ronald R. Yager and Lotfi A. Zadeh. *An Introduction to Fuzzy Logic Applications in Intelligent Systems.* Kluwer Academic Publications, 1992.