Noemon: Adding Intelligence to the Knowledge Discovery Process

Kalousis Alexandros, Zarkadakis George, Theoharis Theoharis
Department of Informatics, University of Athens
grad0058@di.uoa.gr, gzark@hol.gr, theotheo@di.uoa.gr

Abstract. In this paper an architecture is proposed that supports the selection of task, model and algorithm in the knowledge discovery process by utilising artificial intelligence techniques. The proposed system acts as an assistant to the analyst by suggesting possible selections. It improves its data mining performance by observing the analysis sequences applied by the analyst to new data sets. The system associates data characteristics with specific selections that lead to positive or negative results and uses this information to guide later analysis.

1 Introduction

During the last decade people have become increasingly aware of the enormous flood of information, which is challenging our data storage capabilities; it is not uncommon nowadays to have databases containing Gigabytes or even Terabytes of data. Some examples of such databases come from the business area e.g. databases from retailing companies, stock exchange databases, or the scientific area e.g. astronomy, genetics and much more. The amount of information contained in such very large databases (VLDB) far exceeds the human processing and understanding capabilities, so a need has emerged for systems which can aid humans in processing and understanding them. The goal of such systems is to acquire useful, valid and new knowledge from data. The field of Knowledge Discovery in Databases (KDD) tries to serve this goal. A definition of the term KDD appears in [Fayy96]:

> "Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data".

A critical term in the above definition, is "process". KDD is a process consisting of

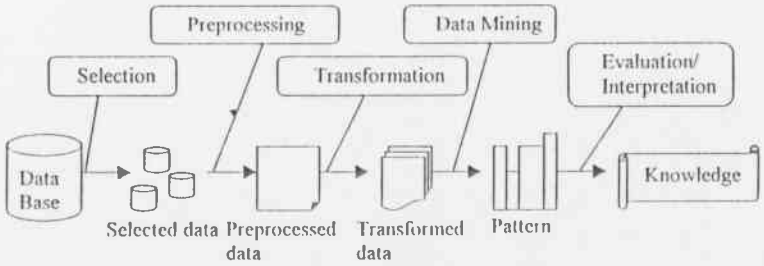

Fig. 1. KDD Stages

several stages, requiring a high degree of analyst-system interaction. In figure 1 [Fayy96] an outline of the various stages of the process is shown.

The stages of KDD comprise:

Selection, where the appropriate data needed for the goal at hand must be collected. This means that the analyst might have to gather data from various sources, perhaps convert them into a common form, and then from this data set select the ones associated with the goal at hand.

Preprocessing where some basic operations on the data take place (e.g. noise and missing field handling), so that they will be ready for use by the algorithms.

Transformation, where either, the data dimensionality is reduced in order to reduce the size of search space, or the data representation is mapped to another space where the solution is easier to find.

Data Mining where the pattern finding is performed, and

Interpretation/ Evaluation, where the findings might be converted into a more comprehensible form, as well as evaluated using criteria that have been established at the beginning of the process.

Of course the whole KDD process is iterative. Whenever the analyst is not content with the results of evaluation or the results of an intermediate stage, he can return at whatever stage he wants. Additionally cases may arise when the process does not evolve in a specific time interval but has a long duration, with new data becoming available that must be accounted for in the knowledge discovery process.

The need for systems that support this highly complicated process in an intelligent way, relieving the user from the routinely performed tasks, is obvious. The architecture we propose herein aims to handle the Data Mining stage of the process in an intelligent way.

In the following section a description of the Data Mining process is given, with emphasis on the problem we are interested in and the existing approaches. Section 3 gives a behavioural and functional description of the proposed architecture (NOEMON), and section 4 presents our conclusions and proposed future work.

2 The Need for an Intelligent Assistant

2.1 The Data Mining Stage of the KDD Process

As discussed in the previous section, the KDD process consists of a number of stages with the analyst having to perform various actions during the process. A description of the Data Mining stage, the stage we are mostly interested in, is given in this section.

Figure 2 presents the steps which, an analyst must follow in order to complete the Data Mining stage. Note that one can conveniently view the various tasks, models and algorithms involved in Data Mining on a 3 level tree architecture.

Firstly, it is the *Task Selection* step, during which the Data Mining Task must be established, based on the problem at hand. The analyst must decide what form of knowledge he is looking for, which in turn will determine the data-mining task. For example, if rules that describe known classes are needed, then we have a classification task; if data are searched for unknown classes we have a clustering task; if dependencies between attributes are searched for we have dependency derivation task.

Let us suppose that the analyst wishes to perform the classification task. Then during the *Model Selection* step, from the various existing models that can accomplish the chosen task, the one that better fits with the data morphology and the analyst requirements, must be selected. For example, if the main concern of the analyst is the understandability of the results, he can choose a decision tree model or a rule model.

The next step is *Algorithm Selection*. The algorithm that provides the best fit of the chosen model to the data is selected, by considering the data morphology. For example if the analyst had chosen decision tree modelling and the classes' boundaries are orthogonal to the axes he can select the c4.5 algorithm [Quin93]. If the classes' boundaries are not orthogonal to the axes he can select OC1 [Murt94] or LMDT [Brod95]. If the data contain noise or missing values then an algorithm that can handle these situations must be selected.

The next step might be the conversion of data in a form that the selected algorithm can handle (*Data Preparation*), e.g. discretization of continuous attributes in case the algorithm handles only categorical attributes; fit the model to the data by applying the algorithm (*Parameter Fitting*), evaluate the results (*Evaluation*) and if he considers they need refinement, return to a previous step of the process or else exit.

2.2 State of the art

There are today a variety of systems offering various models and algorithms (known as multi strategy learning systems) for performing the data mining part of the KDD process, e.g. MLC++ [Koha96], Explora [Klos93] and DBMiner [Han96]. The basic motivation in providing multiple models and algorithms stems from the need for the system to serve different learning goals (i.e. find different forms of knowledge from the data) and to be able to handle data of various morphologies. It has been shown experimentally that each algorithm a *selective superiority*, i.e. it is best for some but not all tasks [Brod93], [Wolp92], [Scha94]. Most systems that comprise a variety of models and algorithms provide little or no guidance as to which one to choose, based on the problem and the data-at-hand. The analyst must consider a variety of factors before deciding which model or algorithm to apply. A good description of the various factors implicated in the model and algorithm selection for the classification task can be found in [Brod97].

The architecture we propose provides support for the first three selection steps i.e. Task Selection, Model Selection and Algorithm Selection. There have been efforts to cope with this problem mainly from the machine learning community. In [Gord95] a framework is proposed for handling the bias selection problem (representational bias and procedural or algorithmic bias), that corresponds to model and algorithm selection. Rendell [Rend87] proposes a similar approach that dynamically selects a representational model and an algorithm to apply for model generation, based on data characteristics. They have implemented a system to test that model called VBMS. Provost [Prov95] proposed a model for selecting inductive bias of various categories. They built a system called SBS

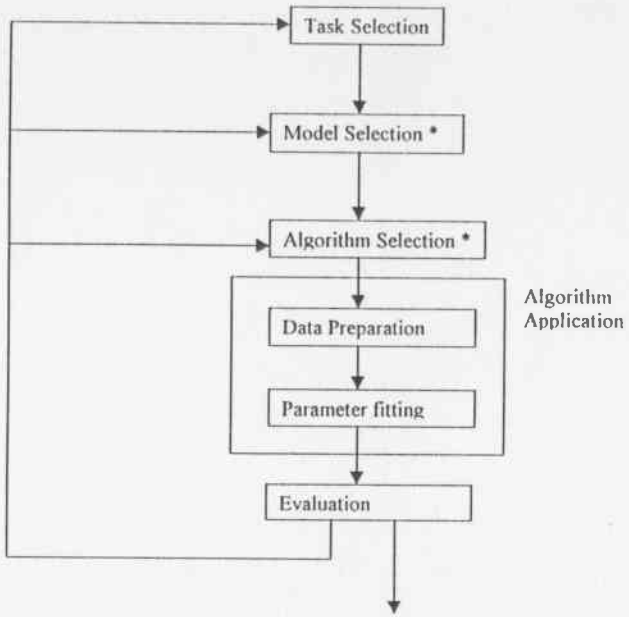

Fig. 2. A description of the steps which an analyst must perform in order to complete the Data Mining stage of the KDD process. The asterisk (*) represents the points where NOEMON will provide assistance to the analyst.

that performed inductive bias selection as a search in the bias space; the main problem of the approach is the high computational expense of the search. In [Brod93] a system called MCS (Model Class Selection) was developed, using rules derived from experts for the automatic selection of a model. The main disadvantage of this approach is that the system can not easily incorporate new models and methods, since the rules were created after thorough experimentation by experts and inserted manually in the system. In [Gama95] a model was built for automatic selection of the best classification algorithm, based on data morphology. Various algorithms were tested on a number of databases and attempts were made to correlate the results of each algorithm with the characteristics of the data. In [Scha93] cross validation was used in order to select the best algorithm. Good results were exhibited but the method is computationally intensive. Another system called AIDE [Aman97] came from the area of exploratory data analysis. In AIDE planning techniques were used in order to provide exploratory data analysis, with strategic abilities, i.e. guidance to the analyst for the selection of the appropriate next step. The disadvantage of this approach is the difficulty involved in incorporating new models or algorithms, since an expert must create new plans in order to handle them.

Having examined the various paradigms one may distinguish systems that support the selection of a model/algorithm into three categories based, on the autonomy they exhibit:

- *Fully automated systems* where the system selects itself the method to apply. Gama [Gama95] is following this approach, but with poor results. Broodley [Brod93] and Provost [Prov95] are also following the same approach.
- *Mixed-Initiative systems* where analyst and system cooperate in order to select the appropriate method. AIDE follows this approach.
- *User guided systems* where the system provides a selection of models and algorithms and the user has the responsibility of selecting the appropriate one in each case.

In our proposal, we believe that a system functioning not completely autonomously but in close collaboration with the analyst, is the best approach in order to add intelligence in the KDD process. Such a system would exploit the merits of intelligent decision-support preserving user action integrity.

Let us briefly examine what such a system would do. During Task Selection, the system should provide hints as to what task is more promising, based on data characteristics. During Model Selection the system should help the analyst select a specific type of model; for example if the analyst chose to perform classification he could use several types of models such as decision trees, nearest neighbours, neural nets etc. Finally during Algorithm Selection, the system should help the user select the appropriate algorithm for model generation. For example, if the analyst chose a decision tree model there is a variety of methods for building it (c4.5, OC1, ID3, LMDT etc). The one that best fits the data should be selected. Finally the system should allow easy incorporation of new models and algorithms, without the need of reprogramming.

3 Architecture description

We propose a mixed initiative system which will act as an assistant to the whole process, by being partially autonomous and -at the same time- respond to user guidance throughout the process. For instance a high level command from the user should trigger the system to make learned suggestions on which strategy to follow, which method to select in order to implement the chosen strategy, which should be the next steps to apply, based on the current results, and so on. The system should also be able to adapt the course

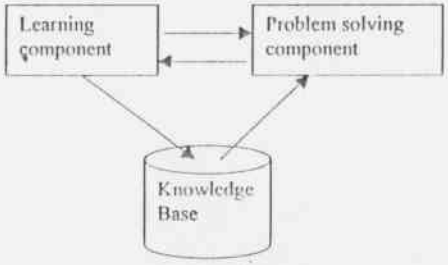

Fig. 3. High level description of the architecture

of its action based on the user's guidance, as well as previous "user history", i.e. averaged user behaviour patterns from previous sessions. In actual fact the system will be able to learn from the user, by observing how he reacts in specific cases, and take similar actions when similar cases arise (learn by example).

The proposed architecture has two components: the learning component and the problem-solving component (figure 3). The problem-solving component performs the actual data analysis and applies models and algorithms in cooperation with the analyst, using knowledge from a knowledge base. The learning component observes the problem-solving component, where analyst and system interact performing the data analysis, and composes new rules. The learning component is therefore used to improve the performance of the problem-solving component.

In the following subsections a description of our system will be given based on the framework that was proposed by Plaza [Plaz93], in order to describe systems that integrate learning into their architecture. Two levels of description will be given:

- *Behavioural description:* a description of *what* the system does, i.e. how the system behaves when we observe its operation.
- *Functional description:* this is the actual architecture of the system, and corresponds to a description of *how* the system does what it does. An explanation is given as to how the system achieves its goals through its knowledge and methods, by means of the specific functions it executes.

3.1 Behavioural description

The system's behavioural description is given in figure 4, by means of a state transition diagram. Boxes represent the various states of the system, arrows represent state transitions and comments on arrows state the event that causes a transition.

At each state there is high interaction between analyst and system. Whatever decisions the system takes are presented to the analyst for approval. He can accept the system's suggestions, ask for an alternative or guide the system explicitly. At each of the three selection states (Task Selection, Model Selection and Algorithm Selection) the analyst can perform the following actions:

1) *Ask for suggestion:* The analyst allows the system to suggest the next step of the analysis. The system comes up with a suggestion and presents it to him. He then has the following options:
   a) Alternative: The system's suggestion does not satisfy him, so he asks for an alternative. The system proposes one, which is again subjected to the analyst for approval.
   b) Accept: The analyst approves system's suggestion and the system proceeds to the next state.
   c) Go back: The analyst is not satisfied with the current state of analysis and commands the system to go back to a previous state and repeat the process.
2) *Specify a selection:* The analyst selects a specific task, model or algorithm. The system accepts that selection and continues to the next state.

The last state of the transition-state diagram is Algorithm Execution. Here the selected algorithm is applied and a model is generated. The analyst evaluates the results and has the following options:

a) *Accept*: The analyst is satisfied with the results of the algorithm and the process ends successfully here.
b) *Go back*: The evaluation results are poor, indicating a poor selection of model or task (e.g. patterns of the form we seek cannot be found in our data) or algorithm. The analyst can return to a previous selection state.

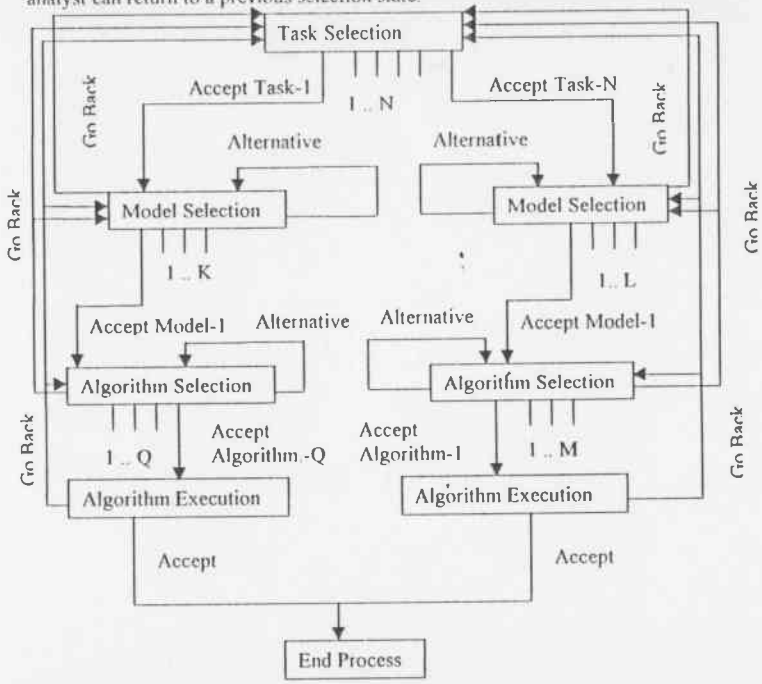

Fig. 4. State Transition Diagram of NOEMON depicting the behavioral description of the system.

The system keeps a log of all interactions with the analyst, at every state of the process. This log is in fact a graph that describes the sequence of events and actions that took place. The graph is used by the learning component in order to improve the system's performance, as described in the following section.

3.2 Functional Description

The system architecture, which implements the behavioural description given in the previous section, is presented here. Figure 5 shows the major components of the system.

The system receives the user requirements and at the same time extracts some characteristic measures from the data. This information is then used to select a task, model and algorithm that appears most promising; the KB rules are applied for this purpose. The selection is then presented to the analyst, who can either approve or reject it as was shown in the previous section. The functional components of the system are described bellow.

3.2.1 HCI (Human Computer Interaction)

This component handles all system-analyst interactions. It gathers the analyst's requirements for the DM process and its results (e.g. process speed, result accuracy, comprehensibility of the final model). These, along with data characteristics gathered from the CE (Characteristics Extraction) component, will be used in the various selections. At each selection state, it presents to the analyst the system's suggestion for the next analysis step. The results of those interactions are sent to the DEC (DECision) component. HCI presents the results of algorithm application to the analyst, who evaluates them and interactively indicates whether he is satisfied with the results or not. This information is passed on to the DEC component.

3.2.2 DEC (DECision)

The DEC component performs the actual problem solving. Its main function is the selection of the next step in the analysis process. DEC sends next step suggestions to the analyst via HCI. If the analyst approves a suggestion the analysis proceeds to the next level, or if we are at the algorithm selection level, the actual algorithm is executed. This is done by sending a message to the Algorithms component, which is then responsible for the actual algorithm execution.

In making suggestions, the DEC component uses two kinds of input. One is analyst input, describing the analyst's requirements. The other input comes from data itself and consists of measures that describe the nature of the data. These measures are:

- Attribute nature (continuous, ordinal, nominal)
- Number of attributes
- Size of sample
- Ratio of the above two quantities.
- Number of classes (on classification task)
- Form of class variable (on classification task)
- Noise
- Attribute correlation
- Dependency Indications
- Statistical, Informational and Entropy measures.

DEC searches the KB in order to find the rules that best match the input at hand. It may be the case that more than one rule applies to the current situation. DEC chooses the rule with the greater belief value; the other rules are stored, in case the analyst rejects that selection or decides at some later step of the analysis to return and get an alternative suggestion.

3.2.3 CE (Characteristics Extraction)

This component is responsible for the extraction of measurements that describe the data morphology. These measurements are calculated when the analyst specifies the part of the database on which he wants the analysis to take place. The results are sent to the DEC component where they will be used as described above. The measurements that are computed are mentioned in the DEC component description.

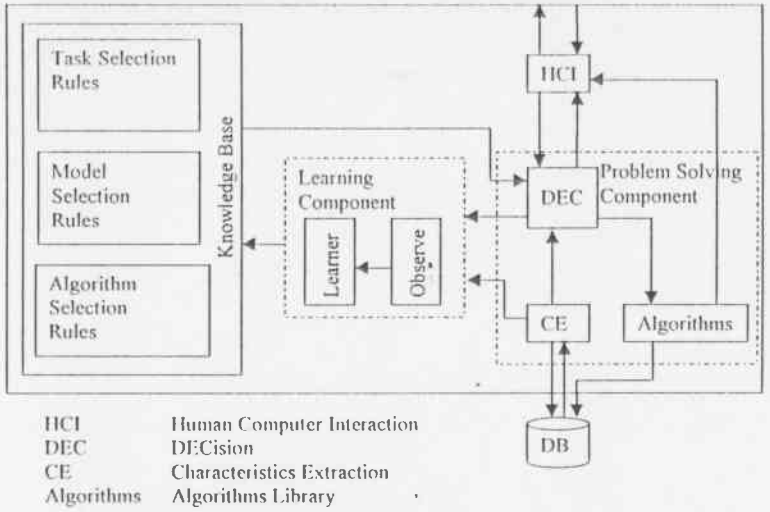

Fig. 5. Functional Description of the System

3.2.4 Algorithms Library

This is responsible for the execution of the algorithms. It receives instructions from the DEC component as to which algorithm to execute on what data. It executes the algorithm and sends the results to the HCI for user evaluation.

3.2.5 Learning Component

This component is responsible for improving the performance of the problem solving part. It continuously observes system-analyst interaction and attempts to introduce new rules which will lead to more successful analysis, or rules that recognise when a certain kind of analysis will not be successful. It consists of two parts. An agent called Observer that gathers data from system-analyst interactions and a component called Learner that is responsible for knowledge acquisition from the information that the Observer gathers.

3.2.5.1 Observer

As mentioned above this component is actually an agent that creates a log of system events. It records all analyst-system interactions, all the suggestions that the system presents to the analyst and his reactions to those suggestions. It notes when the paths

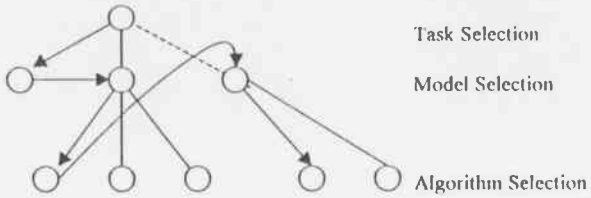

Fig. 6. An example of a graph depicting analyst-system interaction. The descendants of a node are all the possible selections. The children of a node are sorted by their degree of belief. The left child of a node is therefore the system's suggestion. Arrows indicate the path that the analysis follows. Dotted lines indicate new selections, specified by the analyst.

taken lead to failure and how the analyst overcomes this situation. It also notes the paths that lead to successful analysis. All this information is recorded using graph data structures.

The graph consists of three levels, each one corresponding to a selection state. The nodes of each level are the system's suggestions and the alternatives. If the analyst makes a selection which is not one of the system proposed options, a new node is added to the graph (dotted edges). At any point in the analysis, the path followed so far is indicated by arrow headed edges. When a node is selected, links are created to descendant nodes, which describe all possible alternatives at the next level. Arrow headed edges from nodes to nodes of a previous level indicate backtracking.

3.2.5.2 Learner

The Learner uses the graph constructed by the observer, in order to find situations that lead to successful analysis (corresponding to positive examples), but also situations that lead to failures (negative examples). It must be noted here that the existence of failures is as important as the existence of successes. Based on this, the learner updates the rules of the KB.

When certain decisions are made on regular basis, under specific analyst requirements and data characteristics, rules can be created to associate those characteristics with the decisions taken. If the application of a rules leads to successful analysis its belief value is increased. Situations where the analyst selects a different next step from the one the system proposes are also exploited; in this case the belief value of the rule that the system proposed remains the same, and a new rule is created describing the action taken. The only case that the belief value of a rule is decreased is when its application produces different results than the ones expected (i.e. failure instead of success of analysis). The degree of belief of a rule is actually the percentage of times that the rule was successfully used, (i.e. #successful applications / #total applications).

3.2.6 Knowledge Base (KB)

KB provides the information for the successful guidance of the analysis process. The rules it contains can be divided into three categories: Task Selection, Model Selection and

Algorithm Selection rules; each category is applicable to the corresponding analysis phase.

Task Selection rules use data indicators (e.g. attribute correlation, attribute dependencies indications) to suggest a task.

```
TASK SELECTION LEVEL
    Rule   : If correlation between variables
             Suggest REGRESSION
    Rule   : If dependecy_indication Suggest DEPENDENCY DERIVATION
    Rule   : If user asks prediction of variable X and X continuous
             Suggest REGRESSION
             Else Suggest CLASSIFICATION
    Rule   : If user knows the classes
             Suggest CLASSIFICATION
             Else suggest CLUSTERING

MODEL SELECTION LEVEL
    Rule   : If instances linear separable suggest a linear fit
    Rule   : If num_of_instances < 2*num_of_attributes
             Suggest DESICION_TREES or NEAREST_NEIG
             Else
             Suggest LINEAR FIT
    Rule   : If irellevant_attributes
             Suggest DESICION_TREES
    Rule   : If concept_form continuous
             Suggest LINEAR REGRESSION
    Rule   : If understandability
             Suggest DECISION_TREES

ALGORITHM SELECTION LEVEL
    Rule   : If num_of_instances < K and
             prototypes per class > 5.5 suggest X algo.
    Rule   : If cost < K and entropy_of_attributes < L suggest S algo
```

Fig. 7. Examples of Knowledge Base rules

Model Selection rules describe which models are associated with the selected task and which are the most promising in the current situation based on data morphology, user requirements and other parameters.

Finally, Algorithm Selection rules associate algorithms with specific models and describe the situations (data morphology and user requirements) under which a specific algorithm is better than others which implement the selected model. In figure 7 we can see how such a KB would look like.

Each rule is associated with a degree of belief that derives from the percentage of times that the rule performed correctly. This degree of belief changes as the analyst uses the system and the system improves its knowledge, increasing when the rule is successfully applied and decreasing when it is unsuccessfully applied.

A question that naturally arises is how the initial form of the KB is created, so that the analyst will not have to wait for the system to be trained; moreover, by what means one may initially acquire knowledge associated with new models and algorithms. The goal here is to reduce as much as possible the need for an expert who will insert the rules manually to the KB of the system. Some of the initial rules will be entered in the KB by the system developers. However, a non-expert user should be able to insert new models or algorithms without difficulties. Let us now see how new rules may be generated (figure 8).

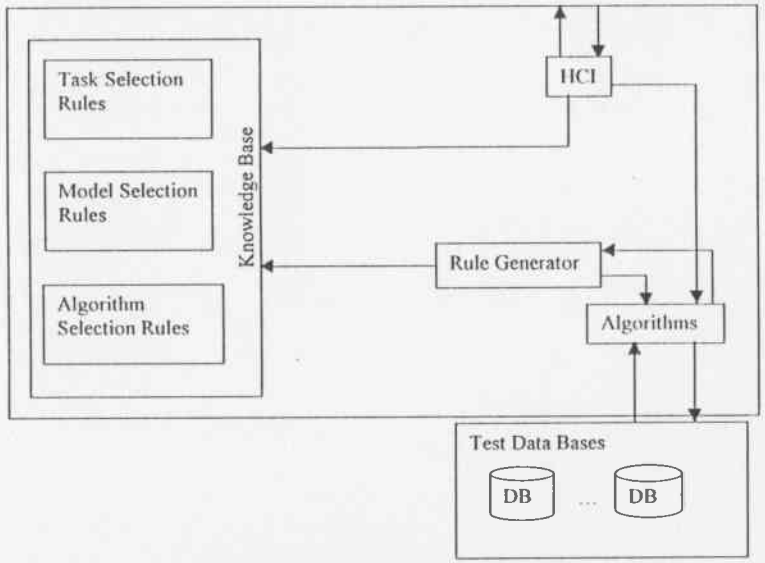

Fig. 8. Functional description of Rule Generator

Task Selection rules are inserted during the development phase. As new tasks will not be inserted anymore (i.e. they are fixed), the analyst need not be concerned with this type of rule.

A new Model Selection rule is inserted along with every model introduced to the system, associating the model with a specific task. At the beginning, the system will not be able to efficiently handle the new model, but this will change in the course of time, once the system builds new rules that guide the model's application.

Similarly, a new Algorithm Selection rule is inserted for every algorithm introduced to the system, associating the algorithm with a specific model. The *Rule Generator* component of the system will build a set of initial rules that will guide the application of the specific algorithm based on the work of Gama [Gama95]. The system will have at its disposal a collection of test databases, each one having characteristic data morphology. The new algorithm will be applied to those databases and the results will be evaluated by the system. Using the morphology of the test databases and the evaluation results, the system produces the initial set of rules, which will associate the data morphology with the algorithm performance. Of course, this set of rules will be refined in the course of time, as the algorithm is applied to new data sets and new knowledge becomes available. It must be noted here that the selection of the test databases is crucial for the construction of the rules that will initially guide the application of the new algorithms. In order for the system to be able to propose analysis steps for various data morphologies the set of test data bases must contain as many diverse morphologies as possible. That way the rules produced for each new algorithm will be more general.

The analyst can also form his own set of test data bases, from data bases used in past analysis sequences; the rule generator performs the initial training on this test set. Apart from that, he can guide the training of the system by presenting it with past analysis cases, taking care to present not only successful but also unsuccessful ones.

4 Conclusions

This paper presented "Noemon", an architecture for the support of task, model and algorithm selection in the KDD process. Previous approaches with similar goals are either static (i.e. they can not incorporate new models or algorithms unless they were reprogrammed), or, in the case of dynamic approaches, they either exhibit poor results or are computationally intensive. We propose an architecture that can incorporate new models and algorithms and learn to use them efficiently in the course of time, by observing the cases where the new models and algorithms seem to perform best. However it must be noted that this architecture learns to handle better the data morphologies that the analyst works more often with. It also tends to adapt to the analyst's personal preferences.

Future work on "Noemon" will include the implementation of the automatic creation of the initial rules that guide the application of a newly inserted model or algorithm, as well as the implementation of the Learning Component. We will first focus our effort to models and algorithms that are used for the classification task. We plan to develop a matrix-laced structure of co-operating intelligent agents that will support the functionality described above.

Acknowledgements

The authors would like to acknowledge financial support from the Greek General Secretariat for Research & Technology (HENEA 28).

References

[Aman97] Amant R. S., Cohen P. R., 1997. Interaction With a Mixed Initiative System for Exploratory Data Analysis. In proceedings of the Third International Conference on Intelligent User Interfaces, pp 15-22.

[Brod93] Brodley C. E., Utgoff P. E., 1993. Addressing the selective superiority problem: Automatic algorithm/model class selection. In proceedings of the Tenth International Conference on Machine Learning, Amherst, pp 17-24.

[Brod95] Brodley C. E., Utgoff P. E., 1995. Multivariate Decision Trees. Machine Learning vol. 19.

[Brod97] Brodley C., Smyth P., 1997. Applying Classification Algorithms in Practice. To appear in Statistics and Computing.

[Fayy96] Fayyad U.M., Piatetsky-Shapiro G., Smyth P., 1996. From Data Mining to Knowledge Discovery: An Overview. In Advances in Knowledge Discovery and Data Mining, pp 1-34. (eds) Fayyad U.M., Piatetsky-Shapiro G., Smyth P. Uthurusamy. AAAI Press/MIT Press.

[Gama95] Gama J., Brazdil P., 1995. Characterization of Classification Algorithms. In proceedings of Progress in AI, 7th Portugese Conference in AI, EPIA 95,pp 83-102. (eds) Pinto Fereira C., Mamede N., Springer Verlag.

[Gord95] Gordon F.G., desJardin M., 1995. Evaluation and Selection of Biases in Machine Learning. Machine Learning 20: 5-22.

[Han96] Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R. 1996. DBMiner: A System for Mining Knowledge in Large Relational Databases. In proceedings of International Conference on Data Mining and Knowledge Discovery (KDD 96), pp 250-255, Portland, Oregon.

[Klos93] Klosgen W., 1993. Problems for KDD and their treatment in Statistics Interpreter Explora. In International Journal of Intelligent Systems 7(7):649-673

[Koha96] Kohavi R., 1996. Data Mining using MLC++ a Machine Learning Library in C++. In Tools with AI 1996.

[Murt94] Murthy S. K., Kasif S., Salzberg S., 1994. A system form Induction of Obllique Decision Trees. Journal of Artificial Intelligence Research 2:1-32.

[Plaz93] Plaza E., Aamodt A., Ram A., van de Velde W, van Someren M., 1993. Integrated Learning Architectures. In Lecture Notes In AI, pp 429-441. Springer 667.

[Prov95] Provost F. J., Buchanan B., G., 1995. Inductive Policy: The pragmatics of bias selection. Machine Learning 20:35-61.

[Quin93] Quinlan J. R., 1993. C4.5 Programs For Machine Learning. Morgan Kaufman.

[Rend87] Rendell L., Seshu R., Tcheng D., 1987. Layered Concept Learning and Dynamically Variable Bias Management. In proceedings of the Tenth International Joint Conference on Artificial Intelligence '87, pp 308-314.

[Scha93] Schaffer C., 1993. Selecting a classification method by cross-validation. Machine Learning 13:135-143.

[Scha94] Schaffer C., 1994. A conservation law for generalization performance. In proceedings of the Eleventh International Conference on Machine Learning, pp 198-202. Morgan Kaufman Publishers, San Mateo, CA.

[Wolp92] Wolpert S., 1992. On the connection between in sample testing and generalization error. Complex Systems 6:47-94.