Gained knowledge exchange and analysis for meta-learning

Norbert Jankowski and Krzysztof Grąbczewski
Department of Informatics
Nicolaus Copernicus University
Toruń, Poland
http://www.is.umk.pl/
{norbert|kgrabcze}@is.umk.pl

***Abstract*— Building accurate and reliable complex machines is not trivial (but necessary in most real life problems). Typical ensembles are often unsatisfactory. Meta-learning techniques can be much more powerful in composing optimal or close to optimal solutions to given tasks.**

Efficient meta-learning is possible only within a versatile and flexible data mining framework providing uniform procedures for dealing with different kinds of methods and tools for thorough analysis of learning processes and their results.

We propose a methodology for information exchange between machines of different abstraction levels. Inter-machine communication is based on uniform representation of gained knowledge. Implemented in a general data mining framework, it provides tools for sophisticated analysis of adaptive processes of heterogeneous machines. The resulting meta-knowledge is a brilliant information source for further meta-learning.

I. Introduction

Most real life classification (and other data mining) tasks are so hard, that single simple classifiers are very unlikely to successfully solve them. The need for complex machines including different data transformations and classification ensembles is undeniable. Even simple ensemble construction methods often end up with poor results because of different reasons. To obtain high classification scores for different datasets of different kinds, we need tools which facilitate not only construction of classifiers, but also proper data transformation and proper validation of complex structures of machines.

Techniques leading to satisfactory solutions of different problems are the subject of research called *meta-learning*. Although this term has been used in numerous articles in the sense of ranking algorithms according to descending probability of being successful when applied to given data [1], [2] or in the sense of simple ensembles construction, we use it in much broader sense of learning how to learn different tasks. Our meta-learning ideas combine different heuristic search procedures based on knowledge extracted from past learning scenarios with active analysis of the results of application of different methods to given data.

Efficient meta-learning techniques will more and more often present us successful models, which would be very difficult for humans to find, because of their unusual structures. Some of our research has already born the fruits of very high accuracies of our classifiers solving tasks of the NIPS 2003 Feature Selection Challenge[1] [3] and the Handwritten Digit Recognition Competition[2] organized with The Eighth International Conference on Artificial Intelligence and Soft Computing in 2006. The models we found were usually complex model structures consisting of some data transformations like standardization, feature selection, features construction based on principal components analysis and some committees of classifiers. We have also examined some aspects of member-model competence in classification committees [4].

All such meta-learning approaches require large amount of calculations (e.g. to validate the candidate machines) before they point to most attractive solutions. Many candidates must be examined, numerous combinations validated often with different optimization criteria. To make it all possible we need a general data mining system, which efficiently manipulates such complex machines. Such system must provide:

- uniform way of machine configuration and machine creation—the possibilities of adding, configuring, training, testing and removing machines in a standard way, implemented as a set of project management routines in such a way that does not burden the authors of particular machines with the administration efforts,
- uniform access to results of machine learning and tests, so that meta-learning machines do not need much (or even any) knowledge about the specificity of particular machines,
- uniform query system for gathering information from submachines, facilitating versatile and efficient analysis of gathered results.

The mechanisms must be uniform but not too restrictive, i.e. general enough to fit any kind of adaptive machines (also the results of machines which will be constructed in future).

The abstraction of management routines facilitates communication between different machines within the project on appropriate, different levels of abstraction. Dependently on particular needs, general or detailed questions may be asked in a common language without the necessity to know the details of the machines being used. This provides an excellent source

[1] http://www.clopinet.com/isabelle/Projects/NIPS2003/
[2] http://www.icaisc.pcz.czest.pl/competition.htm

of knowledge (meta-knowledge) not only for basic analysis of datasets, but also for advanced meta-learning.

During recent years we have been working on a system, which combines all the advantages mentioned above and offers unique universality and versatile kernel for wide applications in data mining, with special emphasis on advanced meta-learning. There are many data mining systems available on the market (freeware and commercial), but we don't know any, providing so rich general functionality of the kernel and being so suitable for multiple complex machines management.

Section II sketches some ideas of the system with special emphasis on the topics of this article (more information is presented in [5]. Next, in section III, we describe the abstraction of machine results representation. In section IV, the general methodology of exploring the results is presented and illustrated by examples.

II. Fundamental ideas of our data mining system

Our system is a general data mining tool eligible for any computational intelligence applications. It's abstraction is based on generalized definitions of *machine* and *model*. An abstract view of *inputs* and *outputs*, *parameters* and *results* provides general tools for machine management, independent of the kind of the algorithm.

a) Machines and models: In computational intelligence, the term *learning machine* (*learning method*, shortly *machine*) is used to describe an *adaptive algorithm*. A *model* is defined as the final result of application of a machine. Our approach extends these terms to encompass a broader (than usual) range of components, because from the point of view of a general data analysis framework there is no reason to differentiate between the algorithms for classification, approximation or clustering and those for loading data, visualizing some aspects of data, testing classifiers etc.

We define a *model* as a result of application of a *machine* (an algorithm) with some particular *parameters* to particular *input* data. A model is an information carrier—this information may be passed to other models by means of model *outputs* and may be put into a special *results repository*.

Such abstract idea of machine and model fits different algorithms corresponding to different levels of abstraction. It encompasses classifiers, data loaders, visualization techniques, tests like cross-validation, etc., and also a part of a complex algorithm if only we specify its inputs, outputs etc. Such solution is very attractive from the point of view of the efficiency of calculations, because it allows to reuse parts of models instead of performing multiple calculations of the same values.

b) Machine hierarchies: All the machines within a project compose two hierarchies. One is defined by the *input–output relation* and has a form of a Directed Acyclic Graph (DAG). The other is defined by the *parent–child relation* and is also a DAG, and more precisely a tree. The parent–child relation is also called *submachine relation* and results from the fact, that machines are allowed to run other machines and use resulting models for their purposes.

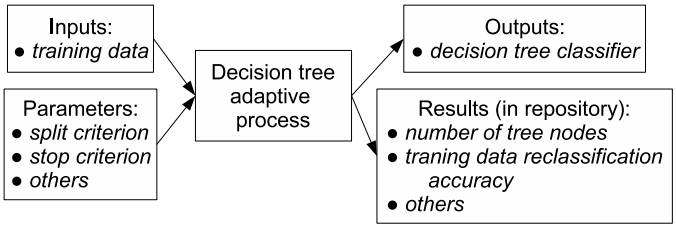

Fig. 1. Decision tree machine structure

c) Inputs vs parameters: The difference between inputs and parameters is that the function of inputs is to provide means for exploiting outputs of other machines, while parameters do not interfere with external machines but specify how the adaptive process of the machine will operate on inputs to generate outputs and results.

d) Outputs vs results repository: The distinction between outputs and results is subtler and concerns the way they can be used by external machines. Both are the effects of the adaptive process of the machine, but while the outputs are to be bound with other machine inputs, the results are deposited in a special repository, which makes them available even after the model itself is released (for example when a vast amount of machine structures is tested, and together they would occupy too much memory). From the other side, the nature of outputs is to provide not only static information about the results, but also methods, to perform the task of the model (e.g. classification), while results repository is rather predestined to contain objects with sort of static information. Another application of the results repository is machine labeling: for example an ensemble can label its submachines to enable easy further analysis of their usefulness for that ensemble, to filter out a group of submachines and calculate some statistics for the group etc.

An example of the scenario with inputs, parameters, outputs and results is shown in figure 1. It shows a decision tree machine with single input of training data and some parameters of the adaptive process. The machine exhibits classification routine as its output and deposits some numbers in the results repository.

e) Machine abstraction: All the ideas mentioned above are general enough to fit any kind of machine (classifier, data loader, classification test, etc.). As a result we created some fundamental classes to implement the common functionality. All the aspects of access to inputs, output exhibition and management, access to submachines and their configurations, results repository navigation etc. have been implemented in *MachineBase* class. The class is common to all possible machines and allows machine implementers to work only on the crucial code for particular machines.

Similar idea stands behind the configuration parameters of the machine. Hence, we have created a general *ConfigBase* class implementing the common functionality related to machine configuration and available to machine developers.

The system has the feature of machine reusability, i.e. if two or more machines of the same configuration are added

to the project, the calculations are performed once, and the resulting machine is put into appropriate contexts. To enable such contexts, each machine is adequately encapsulated (the class which implements this, is called *Capsule*).

III. Results Repository

To satisfy the uniformity of the results management, we created external (to the machine) results repository containing items in the form of *label–value* pairs. The string *label* lets queries recognize the values, which can be *objects* containing information of any type. The object may be a number, string, collection of other values, etc. In most cases objects represent numbers or collections of numbers.

For an example, consider a "classification test" machine. It should certainly have two inputs (a classifier and a dataset to classify) and preferably two outputs (one exhibiting the labels calculated for the elements of the input dataset and one for the confusion matrix). If we want to add the accuracy computed within such model to the results repository, all we have to do is to submit the appropriate label–value pair. To add the value of *accuracy* variable labeled as *"Accuracy"*, we need to call:

machineBase.AddToResultsRepository("Accuracy", accuracy);

where the *machineBase* is the object of *MachineBase* class corresponding to our test machine. Note that each machine and its capsule have their own results repository which is not shared by all machines but is accessible from machines of higher levels (parent machines). Exactly in the same way any other machine can add anything to the results repository, for example SVM can provide the value of its margin, an ensemble can inform about its internals etc.

Another useful possibility (especially for complex machines) is that the addition of label–value items can be done not only in the context of the subject machine, but also in the context of its capsule (which can be seen here as a branch leading to the machine). In this way, the parent machine can add some information about its submachine, which depends on the context in which the child machine occurs, and which the child can not be aware of.

f) Cross-validation example: Consider an example of a well known classifier testing methodology: the 10-fold cross-validation (CV) repeated 10 times. To prepare it, a general machine called *Repeater* may be used, because CV test is just its special case. In general it uses the concept of *distribution boards* and *distributors*. The distribution board is used to generate a number of distributors, and each distributor defines the inputs for a scenario being repeated. In the case of repeated CV test, the *Repeater* uses CV distribution board, which generates 10 pairs of train-test datasets in a random way, each pair is exhibited by a distributor, and is used to perform a single CV fold. The *Repeater* performs the whole procedure 10 times, so as a result, we get 100 submachines. Thanks to the possibility of *labeling* branches (capsules), queries may group submachines by their membership to given repetition or to the *i*-th folds of the CV tests. This significantly enriches the analysis capabilities of the system (see the next section for more details).

To label the branch to chosen submachine a method on its capsule is called:

submachineCaps.AddToResultsRepository("Repetition", repetition);

submachineCaps.AddToResultsRepository("CV-fold", cvFold);

The *object* part of the results repository pair is optional. Sometimes it is enough to use just the label, for example to sign given submachine as a distribution board or a distributor:

distrBoardCaps.AddToResultsRepository("Distribution board");

distributorCaps.AddToResultsRepository("Distributor");

This form is useful also for machines, presenting their results, which have the form of a flag (the same functionality as with boolean type values).

g) Commentators: To extend the functionality of results repository, we have come up with the idea of machine *commentators*. It facilitates extending the information in results repository about particular machines by external entities. The necessity of such solution comes from the fact that the author of the machine can not foresee all the needs of future users of the machine and can not add all interesting information to the results repository. On the other hand it would not be advantageous to add much information to the repository, because of the danger of high memory consumption, which results repository is designed to minimize (in the case of memory-consuming machines they are freed and only the most necessary information is kept in the repository).

Commentators have access to machine's inputs and outputs, machine configuration, and they may also calculate new values. They may extract the knowledge and put it to the repository, from any part of the machine, its neighborhood, or even its submachines.

Commentators can be assigned to machines as elements of the ConfigBase class (see next section for more). The number of results repository items for given machine and the size of objects deposited there are not limited. In order to avoid running out of memory, it is not recommended to deposit too much information in the repository, especially because it is hard to predict what will be really useful in further analysis. An alternative is to use commentators which can be assigned to machines when necessary, saving both memory and computation time.

To better understand the idea of commentators consider the example of classification test machine introduced in section III. By default it deposits to the *results repository* only the accuracy of the input classifier computed for the input dataset, as the most required result. Several other values may also be interesting, and could be deposited in the repository, however to save memory they are not put there by default.

Sometimes, it can be useful to add confusion matrix to the results repository. This can be done very easily because the confusion matrix is already an output of the test machine, so it is available without additional calculations. To add confusion

matrix, a commentator dedicated to output of type *DataTable* can be created. It needs just to send the matrix to the results repository with a chosen label. After adding such commentator to a classifier test machine via a ConfigBase, the confusion matrix will be available to other machines (for example a *Repeater* running a CV test, as a parent of the classification tests and can easily calculate some statistical tests concerning the confusion matrices, after all the child machines were performed and possibly removed from memory. The same type of commentator can be used to add class labels collected as the first output of classifier test. It can be done because these labels are also of the *DataTable* type.

On the basis of the confusion matrix, a few simple and popular factors can be put in the results repository. For example the *sensitivity (recall)*, *precision* and *specificity*. Naturally, they all can be calculated and deposited by a single commentator.

Another useful commentator which can be defined for classification test can be helpful for different statistical tests like McNemar's test. To perform such tests, the information about the correctness of classification of all the instances of tested data is necessary (a vector of boolean values telling whether subsequent classifier decisions were right or wrong). Again, such information can be computed on the basis of the information contained within the classification test machine and its input data (the output labels must be compared one by one to the original class labels provided by the input data). It is very easy to implement this commentator, but its usefulness for statistical analysis may be incredibly high.

The above examples show a small subset of the advantages of results repository and commentators. The functionality makes results manipulation independent of particular machines and provides efficient and versatile access to results of adaptive processes. The ideas are especially valuable for meta-learning, which needs universal and flexible tools for every possible aspect of computational intelligence.

IV. Query system

The aim of the methodology of results repository and commentators is to ensure that all the machines in the project are properly described and ready for further analysis. Gathering adequate results into appropriate collections is the task of the *query system*.

The features of a functional query system include:

- efficient data acquisition from the hierarchy of machines,
- efficient grouping and filtering of the collected items,
- a possibility to determine pairs of corresponding results (for paired t-test etc.),
- a possibility of performing different transformations of the result collections,
- rich set of navigation commands within the results visualization application, including easy machine identification for a result from a collection, navigation from collections to machines and back, easy data grouping and filtering, etc.

The main idea of the query system is that the results repository, which is distributed throughout the project, can be searched according to a query resulting in a collection called *series*, which then can be transformed in a wide spectrum of ways (by special components, which can be added to the system at any time to extend the functionality) providing new results which can be further analyzed and visualized.

All the ideas of *series*, their *transformations* and *queries* are designed as abstract tools, adequate for any type of machines, so that each new component of the system (a classifier, test etc.) can be analyzed in the same way as the others, without the necessity of writing any code implementing the analysis functions.

h) Series: The collection of results obtained from results repository as a result of a query is called *series*. In the system, it is implemented as a general class *Series*, which can collect objects of any type. Typically it consists of a number of information items, each of which, contains a number of values. For example, each item of the series may describe a single machine of the project with the value of its classification accuracy, the number of CV fold in which the machine was created etc. Thus each item is a collection of label–value pairs in the same way as in the case of results repository. Such representation naturally facilitates two main functions of the series: grouping and filtering.

To divide a series into a group of series it is enough to precise the label which is intended to determine to which group the item should belong. For example we may divide a series of classifiers' accuracies into groups corresponding to the corresponding number of CV fold.

Another feature of the *Series* class is the possibility to filter out the items satisfying some condition. To achieve this it is enough to call the filtering routine which is parameterized by label and the filter predicate: the returned series will contain the items for which the value corresponding to the label satisfies the predicate.

i) Series transformations: The series resulting from queries may not correspond right away to what we need. Thus we have introduced the concept of *series transformations*. In general the aim of transformations is to convert a number of input series into a single output series. Some of the most useful transformations seem to be:

- a concatenation of series into one series (e.g. for grouping together the results of two classifiers into a single collection),
- combining two (or more) series of equal length into a single series of items containing the union of label–value pairs from all the items at the same position in the input series,
- calculating the differences of some values describing items within two series (e.g. for the purpose of easy testing of statistical hypotheses like paired t-test).
- calculating statistical tests (t-test, McNemar test, etc.) or some properties (correlations, means, median etc.)

j) Queries: To obtain a series of results collected from results repository, we need to run a *query*. A query is defined by:

- the *root node*, i.e. the node of the project (in fact a machine capsule), which will be treated as the root of the branch of the parent–child tree containing the candidate machines for the query,
- the *condition(-s)* defining which machines of the branch will actually be queried, (usually, and especially when the whole branch is very large, we do not want to query each node of the branch, but only the subset of machines derived from a single configuration object, e.g. the kNN classifiers run within a CV—such restriction is very advantageous from the point of view of computational complexity),
- the *labels* to collect, which correspond to label-value pairs in the results repository.

The result of running a query is the collection of items corresponding to the machines occurring in the branch rooted at the *root node* and satisfying the *condition*. Each of the items is a collection of label–value pairs extracted for the collection of *labels*. For greater usefulness, the labels in the third parameter of the query, are searched not only within the part of results repository corresponding to the queried machine, but also in the description of parent machines (we will see the advantage of such a solution in the following examples).

Consider the example of machine structure presented in figure 2. It is a sketch of the hierarchy of machines obtained with a repeater machine running twice 2-fold CV of two classifiers (in parallel) kNN and SVM. Each labeled box represents a single machine. There are also four groups of classifiers and their tests encircled by unnamed boxes—they are the scenarios defined to run in each fold of the CV. The bullets in the left part of the boxes represent machine inputs, and those at the right side—the outputs. As described before, the repeater machine repeats a sequence of runs resulting from the configuration of the distribution board. The two "Distr Board" boxes correspond to the CV distribution board, which splits its input dataset into 2 parts, preparing it for the CV. The "Distr" boxes are distributors—they use the distribution board output to exhibit proper training and test datasets as their outputs. The repeater created a scenario defined at the configuration stage for each of the distributors. The scenario assumed creation of kNN and SVM machines with inputs bound to proper CV training data and one classification test for each of the two classifiers. The test machines' inputs are bound to corresponding classifiers and CV test data respectively.

After the structure of machines is created and the adaptive processes finished, different queries may explore the results repository. The most desirable query in such case is certainly the query for the collection of CV test classification results of each of the classifiers. To achieve this we may define the query in the following way:

- the *root node* is the repeater node,
- the *condition* is being derived from the TestA or TestB configuration respectively for kNN and SVM,
- the collection of *labels* contains just the "Accuracy" label.

Such query gives us a series of four accuracies calculated by TestA or TestB machines.

If we like to calculate the average accuracies for each repetition of the CV separately, we need the four values to be separated into two groups of two. To do this, we need to include the repetition number in the collection of labels. We are allowed to include as many labels as we like, so lets add the CV fold label too. Both the *Repetition* and the *CV-fold* labels are assigned to the connection leading to the box with classifiers (R1 or R2 and F1 or F2), so appropriate information must be included, but it is too technical to describe it here. As a result we obtain a series of four items consisting of the values of accuracy, repetition number and CV fold number. Now it is enough to call appropriate method of the *Series* class to group the data by the *Repetition* or by the *CV-fold*. We can also filter out the results corresponding to the second repetition of the CV, to the first CV fold etc.

To test the statistical significance of the differences between the results of kNN and SVM with paired t-test (it does not make much sense in the case of 2×2-fold CV, but the way to do the test does not depend on the numbers of repetitions or CV folds), we need to collect the results of all the kNN machines and SVM machines separately (as described above) and transform the two series into the series of accuracy differences with the proper transformation machine, then group the results by the repetition number, calculate standard statistics for each group and use the average and standard deviation in calculations of the paired t-test. Since the standard statistics and paired t-test are implemented as series transformations, the whole process from the query result to the final t-test decision is a sequence of different series transformations. Such design of the query system provides universalism on an unprecedented scale.

Another query may initiate the way to testing statistical significance with the McNemar's test. To run the test procedure we need two series of correctness flags, as described in section III. We need the commentator described there to generate such series for each of the classification test machines. Then the series of these series (obtained for different test machines) may be concatenated with the proper transformation into one long collection for kNN and one for SVM. Please notice that such strategy preserves the correspondence of the results for kNN and SVM at each position of the two resulting series. Thus the McNemar test (also implemented as a series transformation) can be applied to the two series yielding the final result.

Another interesting application of the query system is the analysis of classification test results vs kNN machine parameters. For example, when kNN is configured to automatically select its "k" parameter for given training dataset, we may be interested to see the dependency between test accuracy and the value of "k". To do this we need a series containing items describing both kNN and TestA machines. Thus we may run two queries with the repeater machine capsule as the root node. One of the queries will collect the values of "k" from the kNN machines (the condition should specify that the queried machine must derive from the *ConfigBase* of kNN). The other

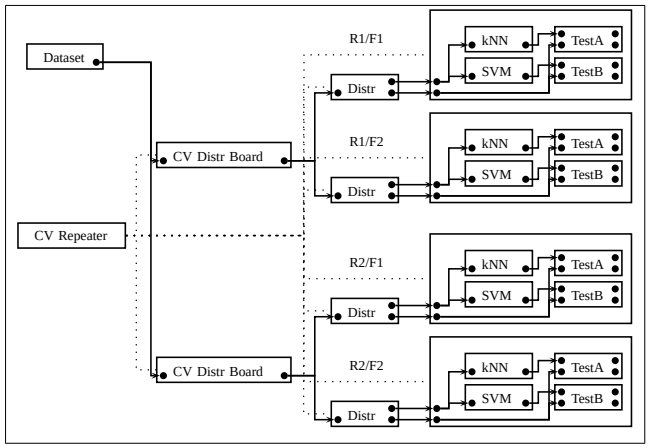

Fig. 2. A repeater machine performed twice a 2-fold cross-validation of two classifiers.

query will analogously collect the accuracies from the results repository corresponding to the TestA machines. Provides the two series (results of running the two queries) we can combine the series with the adequate series transformation, and then group the results by the values of "k". After application of standard statistics for each of the groups we can visualize average accuracies and standard deviations for the "k" values that were selected by the kNN machines.

The number of possible combinations of different commentators, queries and series grouping, filtering and transformations is huge even with a small set of basic commentators and transformations. Since the system is open in the sense that any SDK user can add new commentators and series transformations, the possibilities of results analysis are so rich, that we can claim, they are restricted only by user invention.

The query system has been designed to facilitate advanced meta-learning. It can be successfully used for miscellaneous, sophisticated applications in data mining including construction of different types of classification committees and other ensembles of machines, feature selection and extraction, and many other fields.

Even the simplest tools facilitate high level analysis of validation test results within complicated hierarchies of machines.

To prevent the user from the necessity of defining all details of the commentators, queries and series transformation each time new complex machine configuration is run, the system provides the mechanisms for addition of shortcuts (or templates) which facilitate running similar sequences of operations after specification of the crucial parameters only. For example, in the case of a standard CV test, the query for calculation of basic statistics of classification accuracy is added automatically and may run without any changes in the parameters, although obviously, the parameters may be tuned at any time according to user requirements.

V. Summary

There is no meta-learning without meta-knowledge. This is why we proposed new framework for information exchange in our versatile and efficient data mining system. This framework guaranties the exchange of any information, between machines of any complexity. Universal mechanisms for results repositories services, powerful system of information retrieval from repositories and their manipulation, offer a multitude of possibilities never met in known meta-learning or other data mining systems. The results repositories may contain heterogeneous information. Moreover, the commentators may be used to extract additional information from already implemented machines to extend possibilities of further analysis. Using the system of queries for results repositories, series and series transformers, one can obtain answers for very broad range of questions and successfully mine for meta-knowledge.

All these features cooperate with the rest of our system in perfect harmony.

Acknowledgements: The research is supported by the Polish Ministry of Science with a grant for years 2005–2007.

References

[1] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier, "Meta-learning by landmarking various learning algorithms," in *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, June 2000, pp. 743–750.

[2] P. Brazdil, C. Soares, and J. P. da Costa, "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[3] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, *Feature extraction, foundations and applications*. Springer, 2006.

[4] N. Jankowski and K. Grąbczewski, "Heterogenous committees with competence analysis," in *Fifth International conference on Hybrid Intelligent Systems*, N. Nedjah, L. Mourelle, M. Vellasco, A. Abraham, and M. Köppen, Eds. Brasil, Rio de Janeiro: IEEE, Computer Society, Nov. 2005, pp. 417–422.

[5] K. Grąbczewski and N. Jankowski, "Versatile and efficient meta-learning architecture: Knowledge representation and management in computational intelligence," in *IEEE Symposium Series on Computational Intelligence (SSCI 2007)*, 2007, in print.