Versatile and Efficient Meta-Learning Architecture: Knowledge Representation and Management in Computational Intelligence

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

***Abstract*— There are many data mining systems derived from machine learning, neural network, statistics and other fields. Most of them are dedicated to some particular algorithms or applications. Unfortunately, their architectures are still too naive to provide satisfactory background for advanced meta-learning problems.**

In order to efficiently perform sophisticated meta-level analysis, we need a very versatile, easily expandable system (in many independent aspects), which uniformly deals with different kinds of models and models with very complex structures of models (not only committees but also much more hierarchic models). Meta-level techniques must provide mechanisms facilitating optimization of computation time and memory consumption.

This article presents requirements and their motivations for an advanced data mining system, efficient not only in model construction for given data, but also in meta-learning. Some particular solutions to significant problems are presented. The newly proposed advanced meta-learning architecture has been implemented in our new data analysis system.

I. Introduction

Learning from data is getting more and more important as a way of knowledge discovery for many real world problems. Today nearly everything is (or may be) represented in a digital format, hence may be analyzed using computational intelligence methods. Given a problem represented by a data set $\mathscr{D}$ we look for a *model* which seems to be a good solution to the problem. Such model can be seen as a function

$$f_M : \mathscr{X} \to \mathscr{Y} \tag{1}$$

transforming the domain $\mathscr{X}$ into some target space $\mathscr{Y}$. The goal of finding attractive models may be achieved by adaptation of some free parameters (denoted by M). The task may be defined in a number of different ways. Typical tasks belong to groups like classification, approximation, clustering, rule extraction, finding associations etc. For an introduction to learning algorithms see [1], [2], [3], [4], [5], [6].

A *learning method* is a process of adaptation of free parameters. It can be controlled by a group of parameters called its *configuration* and as a result gives a "final" model (regardless of whether the learning was supervised or not). However the model is "final" only from the point of view of given learning method and its configuration. Using different configurations and/or different learning methods, which solve the same kinds of problems, for given data $\mathscr{D}$, usually leads to solutions of different quality and often of statistically significant difference. Typically only a small subset of solutions is satisfactory and we need special measures to compare solutions and select optimal or suboptimal ones. From the other hand, in compliance with the conclusion of *no free lunch theorem*, we can not expect that a single method may be optimal for every data set. This means that to reliably solve problems, we can not restrict our search to application of a *single method*.

The nature of a problem represented by data $\mathscr{D}$ does not need to be very hard. It happens that a simple method may solve a problem very well, nevertheless we have to discover what the "simple method" is and what configuration of that method is appropriate.

The problem is that thousands of articles were devoted to learning methods in computational intelligence and their modifications, while there is no simple answer to the questions which method, how configured and why, we should use to solve a given task. The knowledge presented in these papers does not help us much, when we are faced with a new problem (a new data set). Nontriviality of model selection is evident when browsing the results of NIPS 2003 Challenge in Feature Selection [7], [8] or WCCI Performance Prediction Challenge [9] in 2006. These competitions show that in real applications, optimal solutions must be composed as complex models obtained in atypical ways. This is even more important when solving more difficult problems in text mining or bioinformatics. Then, only when a good cooperation of submodels in a complex one are obtained, we may hope for a reasonable solution. This means that for example before classification we have to prepare some transformation(-s) (and/or ensembles) which play crucial role in further classification.

Some meta-learning approaches [10], [11], [12], [13] base mainly on data characterization techniques (characteristics of data like number of features/vectors/classes, features variances, information measures on features, also from decision trees etc.) or on *landmarking* (methods are ranked on the

basis of simple methods performances before starting the more power consuming ones). Although the projects are really interesting, they still may be done in different ways or at least may be extended in some aspects. The whole space of possible and interesting models is not browsed so thoroughly by the mentioned projects, thereby some types of solutions can not be found with them.

In our approach the term *meta-learning* encompasses the whole complex process of model construction including adjustment of training parameters for different parts of the model hierarchy, construction of hierarchies, combining miscellaneous data transformation methods and different adaptive models, performing model validation and complexity analysis, etc.

Currently such tasks are performed by humans. Our long-range goal is to eliminate human interactivity in the processes and obtain meta-learning algorithms which will outperform human-constructed models. Here we present the framework facilitating dealing with complex models in a simple and efficient manner. Section II explains why currently available systems are not eligible for such advanced meta-learning tasks and section III presents different aspects of our new system.

II. Why the new architecture for meta-learning is indispensable

Data mining software systems available today do not provide satisfactory tools for meta-level model manipulation. Software packages like free Weka, Yale, commercial SPSS Clementine, Ghostminer etc.—see [14], [15] for a comprehensive list—are designed to prepare and validate different computational intelligence models, but they lack most of the features listed below, which are substantial for effective meta-learning. Thereby these systems may be used like calculators in computational intelligence rather than systems which discover models in really automated and autonomous way. Advanced systems for complex model construction and analysis must provide:

- a unified view of most aspects of handling CI models, (including complex model structures) like model construction and a general input–output representation for information exchange between models, which facilitates common manner of models manipulation without much information about the nature of each particular model (e.g. a unified way of dealing with simple and complex submodels must be provided at the level of the system engine),
- easy and uniform access to model parameters; each model must be assisted by its configuration class with a standard way to adjust its fields and a possibility to describe the characteristic of the fields (linear, exponential, etc.), the scopes of sensible values, etc.,
- easy and uniform access to exhaustive browsing of results of training; a repository of model results, providing uniform access to this information, independent of particular models,
- tools for estimation of model *relevance* (according to the goal, it may be accuracy, MSE, MAP or one of many other measures [1], [2], [3], [16]) together with an analysis of reliability, complexity and statistical significance of differences [17] to other, already found, solutions,
- tools for fast and easy on-line definition of some small extensions of the system like new metrics, new feature ranking algorithms etc.,
- model templates for configuration of complex model structures with exchangeable parts, instantiated during meta-learning,
- versatile time and memory management to guarantee optimal usage of the resources (especially when dealing with very complex model hierarchies), also when run on a computer cluster; this includes model cache systems and unification framework preventing from repeated calculations, which are very probable in massive meta-level calculations ('probable' not because of chaotic meta-search but same models can be used as parts of others more complex systems),
- simple and highly versatile Software Development Kit (SDK) for programming system extensions; SDK users should define just the essential parts of their methods with as little code as possible and with no system-specific overhead.

All the ideas mentioned above confirm a strong need for a new system designed for advanced meta-learning approaches which must be very efficient and versatile in several ways and at different levels of abstraction. The next section presents some very important features of the new system which can brake down the barriers of current systems.

III. Versatile and efficient data mining framework

1) Versatility: In the case of a data mining system, versatility means that many different kinds of data sets can be easily analyzed with many different kinds of tools. We would like to analyze "tabular" data, text data, bioinformatic sequences or microarrays, etc. Each data set needs miscellaneous transformations before the final knowledge extraction methods can be applied. The transformations include standardization, feature selection and aggregation, Principal Components Analysis, multidimensional scaling and many others. Knowledge may be extracted from data with different techniques derived from statistics, machine learning, neural networks, etc. and solving different optimization problems (classification, regression, clustering, etc.).

2) System components unification: In practice it is impossible to extend the engine of the framework each time we want to support a new type of data or model. Therefore we need an abstract definition of a model, underlying the variety of entities mentioned above. Before a successful data analysis system is built a thorough analysis is necessary, aiming at unifying the background of miscellaneous data and algorithms representations.

Such unification is necessary not only as a programming technique justified by software engineering, but as a mean leading to broad functionality of the engine, facilitating information interchange between entities of different levels of abstraction and providing techniques for reduction of memory and computational time consumption. Additional advantage of the unified view is the possibility of exploration of the space of models without deep knowledge about the particular elements of the system, which is crucial in advanced meta-learning.

3) Common input–output scheme: One of the most significant aspects of the unification is a general input–output scheme. It is extremely important, that all the components of the framework use the same general language to describe the necessary inputs and available outputs. It allows to combine the components at the user interaction level and releases their authors from foreseeing all their possible applications. The combinations reveal the same input–output facilities as single components, so they may be managed the same way as simple models.

Thanks to the rapid growth of computing power, today's personal computers are fast enough to perform quite complex calculations in minutes. As a result of that, most of the data sets, being currently used for model testing in numerous computational intelligence articles, can be analyzed by multiple tests and take advantage of statistical measures of significance of the differences between different methods results. Despite that, numerous publications miss such thorough analysis of the results. A general data analysis framework must provide some standard methods for statistical significance testing, and reveal open infrastructure for further extensions. In combination with the unified input–output scheme it facilitates effortless, reliable testing and comparison of complex model structures.

4) Results repository: Each component of a data mining project should present its most interesting gains to the system, so that the interactive user or other models can take advantage of them. It becomes especially important when meta-learning algorithms are constructed, so it is worth to introduce a special repository for this kind of information. A uniform results repository makes analysis of model results model-independent and facilitates very deep meta-level analysis with simple means. We can not rely on the data contained within the models, because sometimes it is worth to keep the results available even when the model itself is no longer needed, for example in multiple tests like cross-validation, the validated models must usually be released to save memory, but keeping the most important results allows for additional non-typical analysis after the whole calculations are finished.

5) Software Development Kit (SDK): Successful data analysis framework must be assisted by easy to use tools supporting the development of third-party components. The external developers should not be obliged to learn much of the system internals. Defining the necessary stuff like model configuration, its inputs, outputs and the results to be kept in the repository must be as simple as possible, and obviously clear examples, which can also serve as start points, must be provided.

6) User interface: The ease of navigation within the system can be obtained only if the parameters of the algorithms in an intuitive way. Boxes with clearly marked inputs and outputs, arrows displaying the flow of information and context-dependent way of setting parameters seem to be very adequate here.

7) Other ideas: There are plenty of other problems, which must be solved to get a fully-functional framework, like running under different operating systems, parallel calculations on a group of computers with possibly different hardware parameters and even operating systems, etc. In this article we do not describe them, because we concentrate here on the aspects of information exchange between the components and the methods of saving time and memory.

A. Methods and models

In computational intelligence, the term *method* (or *learning method*) is used to describe *adaptive algorithms*. A *model* can be defined as the final result of application of a method. In practice, the term model, often means a representation of some fully-functional model performing approximation, classification or other tasks (e.g. a neural network, a decision tree, a k Nearest Neighbors model, etc.).

It is very popular to split knowledge acquisition process into stages (including data preprocessing and final model construction). So popular, that most data mining systems make clear distinction between these stages. We observe that it often leads to a misuse of learning strategies (for example a supervised discretization is performed as a data preprocessing, and then method capabilities are evaluated with a cross-validation performed on the discretized data).

We propose a unified view of model without the distinction of data preprocessing and proper model building, because in fact, the border between data transformations and final model is vague and gets completely ambiguous when we exploit meta-learning techniques. In our approach the term model encompasses a broader 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 loading the data, visualizing some aspects of data or other models, testing classifiers or approximators, etc. For example a cross-validation test can be treated as a model, because it also performs some calculations to gather some information—in this case the information about series of results obtained with some adaptive processes. The output it generates can also be an input for other models: for example some algorithms controlling statistical significance of differences between different methods results.

1) Models abstraction: In our approach, a *model* is a result of application of an algorithm with some particular parameters to particular input data. It is an information carrier—this information may be passed to other models by means of model outputs (see figure 1). Such abstract idea of model fits different algorithms corresponding to different levels of abstraction. In other words, the general definition encompasses

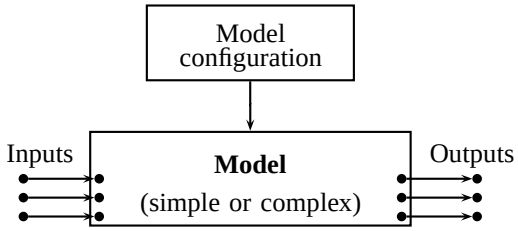

Fig. 1. Abstract view of a model.

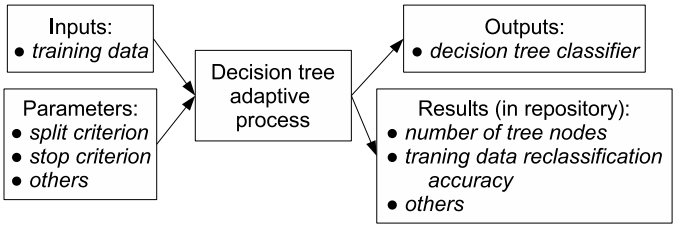

Fig. 2. Decision tree model structure

not only components mentioned above (classifiers, data loaders, visualization techniques, tests like cross-validation, etc.), but also any part of a complex algorithm. For example we can split the SVM methodology into several stages: separate the kernels calculations from quadratic programming tasks etc. In this way we obtain a submodel of SVM, which calculates the kernels—it is a proper model in the sense of our approach, because it precisely defines the input data, kernels parameters, and yields outputs in the form of a table of kernel values. Such solution is very attractive from the point of view of the efficiency of calculations. If we start another adaptive process of the SVM, which does not differ from the first one with respect to the kernels, then the kernel part may be shared between the two SVM models and this way we obtain significant savings in both memory and computational power consumption. The unification of the kernel submodel can be performed automatically by appropriate design of the project management part of the system engine provided that the inputs and parameters of models are also uniform, so can be handled in the same manner on a high level of abstraction.

Also when we use the same data transformation technique as a preprocessing stage for two different learning machines, there is no reason to perform the transformation twice and occupy twice as much memory. If the data transformation is implemented as a separate model, then the model management routines will notice the unification possibility and will use the same transformation model for both algorithms.

Another spectacular example of memory and power savings are the families of feature selection and vector selection methods. We do not need to copy data, when we select a subset of features or vectors. Thanks to submodels extraction we may obtain the same submodel representing the whole data set for each of the models, and the subsets may be defined by sets of indices which usually occupy significantly less memory than the corresponding data subset. Although the access to such selected features or vectors must be a bit more expensive than in the case of copied data, proper definition of the enumerators makes the difference not too large, and savings which result from not copying the data will usually compensate the loss.

2) Inputs vs parameters: One of the ideas mentioned above, that require some additional comment is the distinction between inputs and parameters. Formally the function of inputs is to provide means for exploiting outputs of other models, while parameters do not interfere with external models but specify how the adaptive process of the model will operate on inputs to generate outputs and results.

3) Outputs vs results: The distinction between outputs and results is subtler and concerns the way they can be used by external models. Both are the effects of the adaptive process of the model, but the results are deposited in a special repository, which makes them available even after the model itself is released. From the other side, outputs nature 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. Although the methods of the result objects may also provide extended functionality, it is not recommended to mix the solutions this way.

4) Model structure example: An example of the scenario with inputs, parameters, outputs and results is shown in figure 2. It shows a decision tree model with single input of training data and some parameters of the adaptive process. The model exhibits classification routine as its output and deposits some numbers in the results repository. We can use the output to classify other data sets. This operation makes sense only while the decision tree model is fully available. When the model is tested within a cross-validation, where in order to save memory we do not keep in memory all the models built in each fold, the classification routines of the released models are not available, however it is still possible to analyze the results in the repository, for example to check the numbers of tree nodes obtained in each fold, calculate their averages, standard deviations, etc.

B. Information exchange and complex model structures

The information exchange between models is a crucial feature of an effective system. Indeed almost everything data analysis systems do, is an information exchange or preparation because of information exchange. Separate models are not satisfactory even in the simplest cases. For example, when planning a cross-validation we need some submodels to learn and some others to perform the tests. There are many reasons, for which the possibility of building complex structures of models is obligatory for contemporary data analysis tools.

1) Modular structure: Any model may contain a number of submodels of any type and any level of abstraction. Also a single model may have submodels of different types (for example few feature selection models and few vectors selection models plus one simple committee). The submodels can be seen as *slaves* of the parent model. The submodel does not need to be

of a simple type—it may also be a more or less complex model (e.g. ensemble of complex models, meta-learning, testing model, etc.). Such solution is important in many cases: some typical applications are testing models (repeaters, monte carlo, cross-validation), ensembles, meta-level methods and others. The submodels can be called and used up to the needs of the parent model—the parent model may for example create 1000 models and after that choose three of them and destroy the rest. The important view of submodels cover also the unification level for nontrivial models as it was already presented in the previous subsection by the example of the SVM model which may contain a submodel devoted to the management of the kernels. Because of such definition, models become clearer and much more effective. Such model splits should be performed wherever the adaptive process consists of some naturally separable parts.

2) Input–output interfaces: Models may be connected using input and output interfaces which play the role of plugs and sockets. And as in the world of plugs and sockets they must be compatible (in types and features). The connections are the way of information exchange between models. Output types define exact possibilities of the outputs. It may happen that a single model will have a few different outputs and/or will need a few inputs. Thanks to the inputs and outputs different types of models may be connected to interact (for example clustering model with data loader, classifier with transformer, tester with approximator and data, etc.). Figure 3 presents an example. Dependently on the type of connections, the first model may understand the second one deeper or shallower (according to the needs which always are declared in the specification of inputs).

3) Scheme boxes: Another concept is derived from the nature of flow diagrams. It arose from planning maximum versatility and optimal usage of computational power. A *scheme box* is a specialized type of model to deliver possibilities of enclosing a variety of models and their connections using DAG's (directed acyclic graphs) at the same level of model dependencies. Each pair of models in the scheme box are regarded to be in sibling relation (in contrary to the submodel concept described above, where models are in a parent–child relation). A scheme box may be equipped with inputs and outputs like any other model. Because of that, the scheme may also play the role of a submodel while representing some complex behavior. Scheme inputs may be connected with appropriate model inputs inside the scheme and the same concerns the outputs. The combination of the two concepts of scheme box and submodels allows to build models of any complexity with high efficiency (graphs of graph's).

A good example of how to use the scheme boxes is a model performing cross-validation of classifiers. In our system, it is a specialization of a general model called *repeater*, which is responsible for multiple running of (possibly complex) scenarios. The repeater model is based on the concept of *distribution boards* and *distributors*. This means that each repeater uses an external *distribution board model* to generate inputs for subsequent runs of the repeated procedure. A distribution board is allowed to generate a number of input collections, which are provided to other models by a number of instances of a special model called a *distributor*. Each distribution board defines what distributors may be used with it, so that the repeater can do its job without compatibility clashes. The way, a repeater operates is the following:

- a defined number of times it produces an instance of the distribution board (according to its configuration),
- for each distribution board it generates a number of distributors (according to the information supplied by the board output,
- for each distributor, it constructs a hierarchy of models defined by a scheme box with inputs collection compatible with the distributor.

In the case of repeated CV test, we define the *CV repeater* as a repeater with distribution board fixed to *CV distribution board*, which appropriately generates a given number of pairs of train-test datasets. Each pair of datasets is exhibited by a distributor, and is used to perform a single CV fold.

At the configuration stage the CV model may look like the one in figure 4. The dotted lines connecting the CV Repeater with the CV Distr Board and the scenario scheme box, show the parent–child relation between the models. The CV distribution board has a single input defining the dataset within which the CV is to be performed and a single output providing information about the distribution board (how many distributors are needed and how to create their outputs). The proper scenario, which is to be repeated, is defined as a scheme-box. In this case it uses two inputs (corresponding to the training and test datasets respectively) and allows the user to define its interior i.e. to put there required models and bind their inputs and outputs. A complex structure of models can be generated (including data transformations, classifiers etc.). The example shown in the figure will train two classifiers (kNN ans SVM) in parallel and test each of them in each fold of the CV.

At runtime the CV model acts as standard repeater model (described above). So, it creates given number of CV distribution boards, a number of distributors (equal to the product of the number of repetitions and the number of CV folds) and for each distributor, instantiates the scenario defined within the scheme box. Full view of twice repeated 2-fold CV of the scenario defined in figure 4 is presented in figure 5. Again, the dotted lines show the parent–child relation between the CV Repeater and all of its submodels. Obviously the CV repeater model may also control the results obtained with all of the children, calculate statistics etc.

There are no limits on the types of models that may occur within a scheme box. We can place there different transformers, classifiers, approximators, ensembles, testers, help models, data loaders, etc.

Another advantage of scheme boxes is that they can be used to define templates at model configuration phase (a template of a classifier or other type of model—remember that a scheme with a classifier output may play the same role as other classifiers while having possibility to consist of more than one

Fig. 3. Input and output interfaces. Circles on left and right sides of boxes represent inputs and outputs respectively.

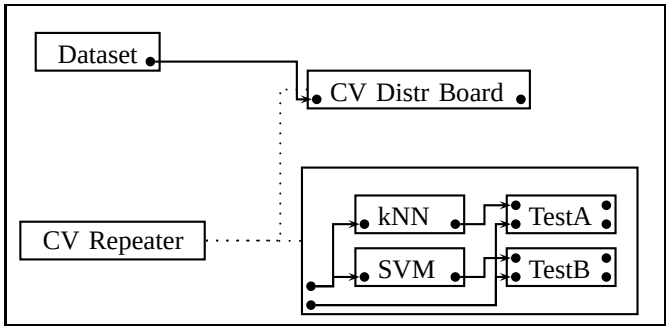

Fig. 4. Configuration of a CV Repeater for classification.

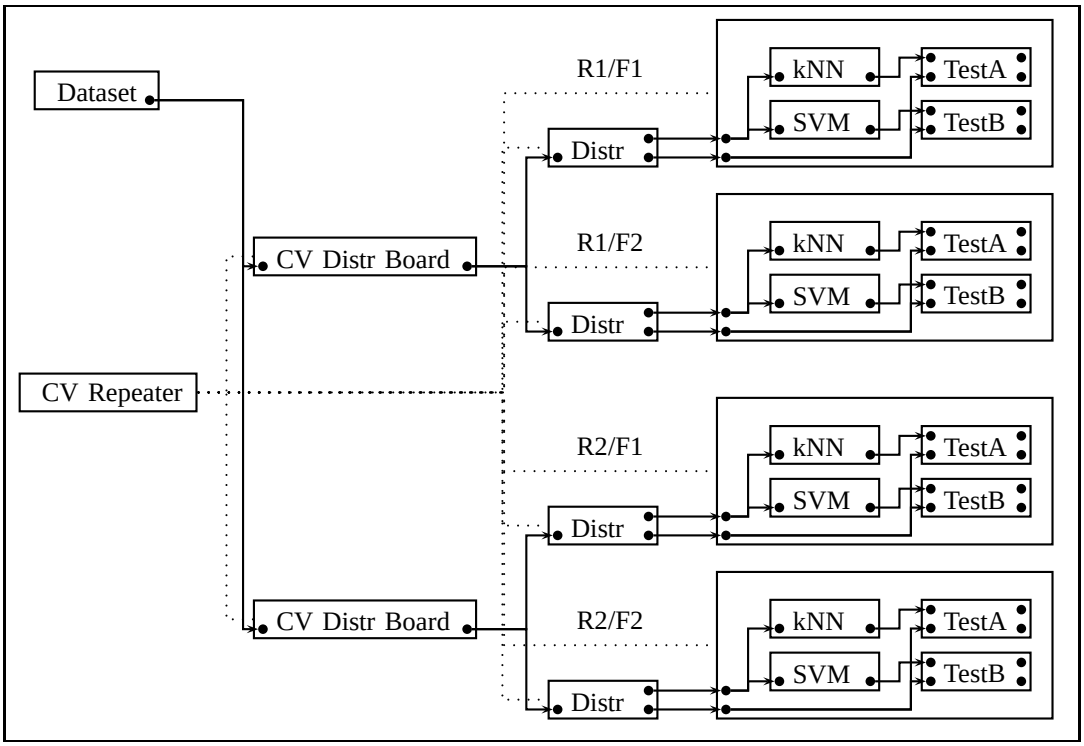

Fig. 5. CV Repeater of figure 4 at runtime.

model). It is especially useful in meta-learning (see section III-C.4 and figure 6).

4) Information exchange using inputs, outputs and results repository: There is a natural correspondence between input and output interfaces. They are responsible for information exchange both in the time of learning and in the further life-time of a model. The connections are determined within model configuration (by the interactive user or enforced by parent models). Models connected to others may open appropriate inputs and read information from them. The structure of the information depends only on the functionality of the input type or more precisely on the source model output type properties which are real objects of concrete types. The type indicates a level of abstraction, for example consider two outputs: *SVM-classifier* and *classifier*. The *classifier* output can be seen as any classifier so it gives access to some general classification routines, however in the lifetime it is a concrete classifier (possibly SVM) but it is of no interest when used for example by an ensemble of classifiers or cross-validation. For some other models it may be very important to read information which is accessible only from *SVM-classifier* output (such output may express additional information related to SVM models). It may be the case when a model has specialized submodels for given purposes. A model may read information

from parent or sibling models via input–output connections. A parent model may read from its child models through known child's outputs. Model outputs are also used by visualization and reporting tools.

Interesting information may be found also in the results repository. The main goal of this repository is to provide a common way for commenting on models which is especially useful for testers. There is no obligation for the models to use the results repository. It is rather an opportunity to present interesting information. Results repository collects information from every model in the project. In fact the repository is distributed according to the structure of the project and it can be read and analyzed in different ways by a special query system. The answers are special objects (with special output types) available as any other outputs, so they may be analyzed by other models (typically by testers, statistical significance analysis methods and especially by meta-learning methods). Results analyzed for example by one of meta-learning model may be again a source of information for another level of abstraction (may be after some pruning if necessary).

There are other levels of unifications in our system which correspond to visualizers, reporting methods, data loading and exporting or working with different types of data. We can not present them all here, because of the space limit.

C. Models manipulation

1) Project manager: Full advantage of the models abstraction described in section III-A.1 can be taken only with appropriate model management. Models are defined and trained within the graph of models, which can be called a project. Providing efficient mechanisms for adding new nodes to the graph, defining input–output connections, etc. are the tasks of a project manager.

In order to efficiently configure models, bind their inputs to compatible outputs, etc. project manager should be equipped with repositories of model types, their inputs, outputs, etc.

2) Models reuse: Supporting models configuration and navigation within the project is an important part of a data analysis system, however the most important part of the project manager seems to be the module for model management, which in particular is responsible for models unification and multiple use of the same component. The repository of all the models in the project augmented by configuration comparison routines can do the job. It is easy to verify whether the inputs of two models are the same. If apart from that each model configuration provides a method to compare two configuration objects, it is easy to recognize when a model can be reused. In conjunction with the model abstraction ideas, which suggest splitting complex models to several more specialized ones (extracting kernels from SVM, building appropriate models for data tables, etc.), the model manager will facilitate reuse of models parts, and therefore will reduce the time and memory consumption.

The model reuse may be much broader, when we supply the system with model cache. The models released from the project may be kept in the cache for some time, and possibly be reused in the future. Different types of cache may be implemented. The simplest one keeps models in memory during a single session (obviously with additional conditions determining when to finally release the model). Another cache module may keep the models in a database stored on a disk, which allows for models reuse among sessions. Yet another cache system could be designed as a network server and provide mechanisms for sharing models by many users of the system. This would allow for reliable comparison of results obtained with different models for popular data sets without the need for recalculating results of all the models used in the comparison.

3) Complex navigation with no consequences for SDK users: It is very important to design the project management module in a manner which does not burden SDK users with the necessity of deep familiarity with the engine mechanisms. To keep new model development as simple as possible, the cycle of model life must be very simple: each model is configured first and then its adaptive process is started. When the learning is finished the model is fixed and will not change in the future—there is no need to implement the ways of reaction to the changes in other models. This is the point of view of a programmer developing models. From the point of view of a user, each model may be reconfigured and trained many times, but in fact, each time a new model is constructed or reused. Thus, it is very important to sensibly split complex models into a set of smaller ones, because this will make submodels reuse more frequent.

Appropriate design of the SDK and basic models available in the system can "enforce" proper models construction by SDK users. For example, in our system, the methods of feature selection based on rankings of features are defined in such a way, that the ranking is an output of a submodel. Adding new ranking based selection to the system consists in creating just the ranking submodel.

4) Template model structures: Especially for the purpose of meta-learning, model schemes may contain abstract boxes—placeholders which are filled with a concrete model or scheme determined in the meta-search process. Figure 6 presents an example of a meta-learning model configuration with such a template. The meta-learning here will search for a transformation (different transformations, which in particular may be complex structures of transformations, will be tried in place of the "Trans. template") maximizing some measure of quality of the collection of classifiers (the scheme output is a multi-output i.e. a collection of classifiers—in this case a collection of three classifiers: "Classifier 1", "Classifier 2", and "Decision module" which combines decisions of the four classifiers). Please notice that transformations "Trans. 1" and the template are shared in a very natural way, saving computational time and memory (in meta-learning taking care of as small complexity as possible is especially important).

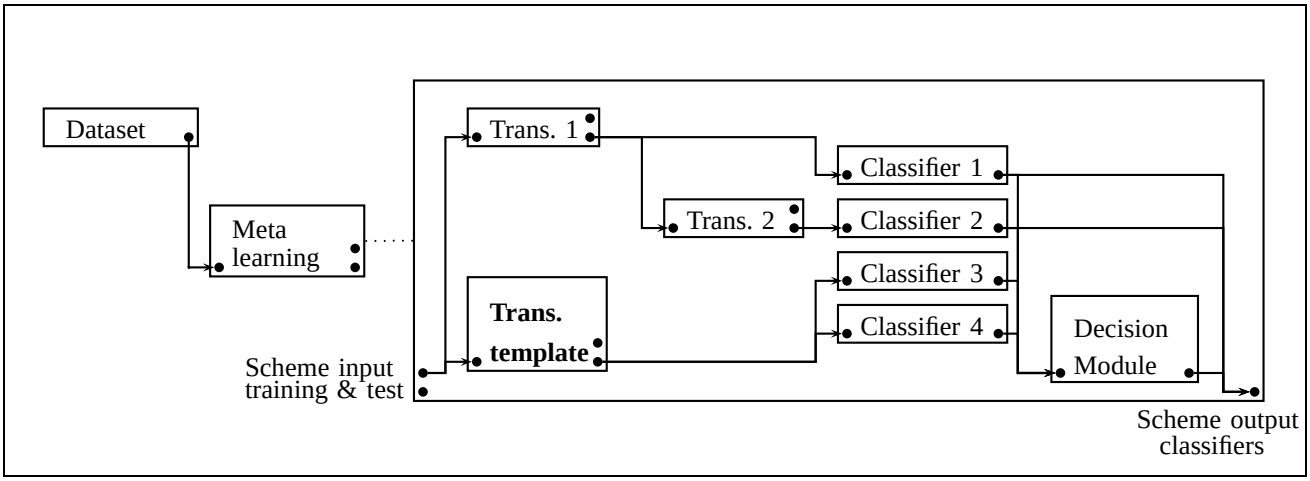

Fig. 6. An example of meta-learning model with transformation template.

IV. Summary

The system, we present the part of here, is very general but still highly effective. Thanks to its modular structure nothing must be reimplemented or recalculated. The possibilities of building models of any complexity facilitates any compositions of known methods (and methods that will be available in the future too).

Thanks to general and flexible engine, new models (also the complex ones) can be implemented effectively with the SDK. Moreover, by means of SDK any type of models can be implemented (classifiers, approximators, testers, measures and even models of completely new types already unknown). Models in the project are connected using input and output interfaces in a natural way giving the opportunity to efficiently exchange information, and the results repository collects some additional data (comments) about the models. The project may contain any number of data sources, any number of simple or complex models of any kinds, which can cooperate or coexists in several ways. Such models may easily exchange information on different levels of abstraction. The versatility of the system predestines it to a broad range of applications including the most sophisticated ones like advanced meta-learning approaches. The riches of different models and their types opens the gates to powerful exploration and explanation of data and can not be compared to any already existing system.

Acknowledgement

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

References

[1] N. Jankowski and K. Grąbczewski, "Learning machines," in *Feature extraction, foundations and applications*, I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Eds. Springer, 2006, pp. 29–64.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley, 2001.

[3] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer Series in Statistics. Springer, 2001.

[5] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 1996.

[6] B. Schölkopf and A. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[7] I. Guyon, "Nips 2003 workshop on feature extraction," http://www.-clopinet.com/isabelle/Projects/NIPS2003/, Dec. 2003.

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

[9] I. Guyon, "Performance prediction challenge," http://www.modelselect.inf.ethz.ch/, July 2006.

[10] 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.

[11] 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.

[12] H. Bensusan, C. Giraud-Carrier, and C. J. Kennedy, "A higher-order approach to meta-learning," in *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, J. Cussens and A. Frisch, Eds., 2000, pp. 33–42. [Online]. Available: citeseer.ist.psu.edu/article/bensusan00higherorder.html

[13] Y.H., Peng, P. Falch, C. Soares, and P. Brazdil, "Improved dataset characterisation for meta-learning," in *The 5th International Conference on Discovery Science*. Luebeck, Germany: Springer-Verlag, Jan. 2002, pp. 141–152.

[14] W. Duch, "Software and datasets," http://www.phys.uni.torun.pl/˜duch/software.html, 2006.

[15] KDnuggets, "Software suites for Data Mining and Knowledge Discovery," http://www.kdnuggets.com/software/suites.html.

[16] T. Mitchell, *Machine learning*. McGraw Hill, 1997.

[17] R. Lowry, "Concepts and applications of inferential statistics," http://faculty.vassar.edu/lowry/webtext.html, Vassar College, Poughkeepsie, NY, USA, 2005.