Building Meta-learning Algorithms Basing on Search Controlled by Machine Complexity

Norbert Jankowski and Krzysztof Grąbczewski

***Abstract*— Meta-learning helps us find solutions to computational intelligence (CI) challenges in automated way. Meta-learning algorithm presented in this paper is universal and may be applied to any type of CI problems. The novelty of our proposal lies in complexity controlled testing combined with very useful learning machines generators. The simplest and the best solutions are strongly preferred and are explored earlier. The learning algorithm is augmented by meta-knowledge repository which accumulates information about progress of the search through the space of candidate solutions. The approach facilitates using human experts knowledge to restrict the search space and provide goal definition, gaining meta-knowledge in an automated manner.**

I. Introduction

META-LEARNING is learning how to learn. In order to perform meta-level analysis of *learning from data* one needs a robust system for different kinds of learning with uniform management of miscellaneous learning machines and their results. Our data mining system is an implementation of a very general view of learning machines and models. Therefore it is very flexible and eligible for sophisticated meta-level analysis of learning processes [1], [2], [3], [4].

Some meta-learning approaches [5], [6], [7], [8] are based on data characterization techniques (characteristics of data like the number of features/vectors/classes, features variances, information measures on features, also from decision trees etc.) or on *landmarking* (machines are ranked on the basis of simple machines 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 other adaptive processes, performing model validation and complexity analysis, etc.

Currently such tasks are usually 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.

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

The pursuit for the optimal model, when performed by a human expert, is usually a search in the space of models, restricted by the experts knowledge of what combinations of techniques are worth a try and which are not. The candidate models are tested in the order determined by the expert. The goal of our meta-learning approach is to mimic this process with automated tools.

Nontriviality of model selection is evident when browsing the results of NIPS 2003 Challenge in Feature Selection [9], [10] or WCCI Performance Prediction Challenge [11] in 2006.

In the case of human experts the order of the tests is based on the experts experience which sometimes can be formally described and sometimes is just some kind of intuition. In computational intelligence the criteria must be precisely defined and thus we introduce our definition of machine complexity, inspired by Levin complexity, which reflects both structural complexity of resulting models and the time of computations necessary to obtain the results.

According to the well known rule called Occam's razor, the simplest machines should be tried first and more complex ones used only if the simple ones do not provide a satisfactory solution to the problem. By complex learning machines we mean both complicated hierarchies of learning algorithms and the processes that are very time consuming.

In this article we present some details of the machine complexity based search which constitutes our meta-learning algorithm. It is an efficient algorithm, which can find many interesting solutions and is a good starting point to even better procedures, which will be certainly created as further steps of our research, because our general data mining platform opens the gates to easy implementation of advanced meta-learning techniques, gathering and exploiting meta-knowledge.

II. General meta-learning algorithm

The major distinction between meta-learning algorithms (MLA) and learning algorithms is that meta-learning concludes and learns from *observations* of single learning processes (or their subparts). It can be seen as additional level of abstraction in the adaptive algorithms. Basing on such observations and on the behavior of several meta-learning algorithms, we propose the general meta-learning algorithm presented in figure 1.

The meta-learning algorithm, after some initialization, starts the main loop, which up to the given *stop condition*, runs different learning processes, *observes* them and *concludes* from their gains. In each repetition it defines a

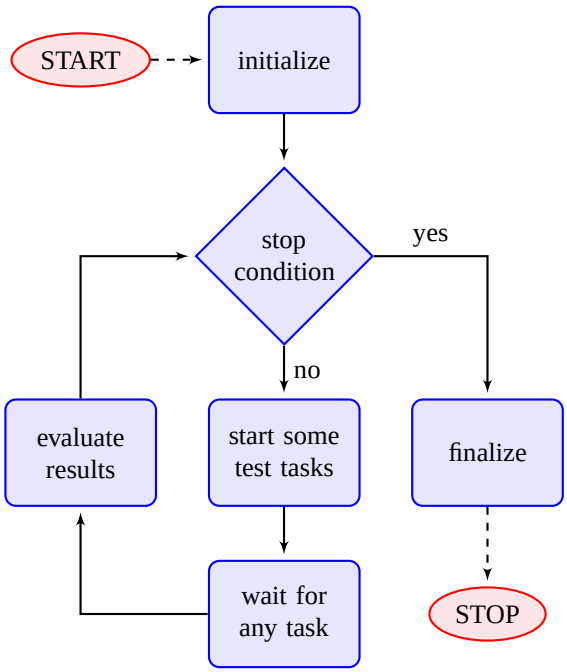

Fig. 1. General meta-learning algorithm.

number of tasks which test behavior of appropriate learning configurations (i.e. configurations of single or complex learning machines)—step *start some tasks*. In other words, at this step it is decided, which machines are tested and how it is done (the strategy of given MLA). In the next step (*wait for any task*) the MLA waits until any test task is finished, so that the main loop may be continued. A test task may finish in natural way (at the assumed end of the task) or due to some exception (different types of errors, broken because of exceeded time limit and so on). After a task is finished, its results are analyzed and *evaluated*. In this step some results may be accumulated (for example saving information about best machines) and new knowledge items created (e.g. about different machines cooperations). Such knowledge may have crucial influence on further part of the meta-learning (tasks formulation and the control of the search through the space of learning machines). Precious conclusions may be drawn, even if a task is finished in a non-natural way.

When the *stop condition* becomes satisfied, the MLA prepares and returns the final results like a ranking of learning machines (ordered by a degree of goal satisfaction), comments on chosen learning machines and their interaction, etc.

Each of the key steps of this general meta-learning algorithm may be realized in different ways yielding different meta-learning algorithms. The following sections present more details of the MLA based on search controlled by machines complexity, but first, some aspects of machine definition are presented.

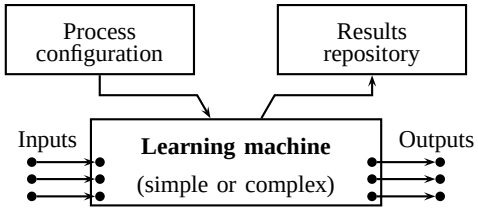

Fig. 2. Abstract view of an adaptive process.

III. Machines, submachines, schemes and information flow

According to our abstract view of learning, a *learning machine* is any adaptive algorithm. It may get some data as input and as a result of the process we get some output. A general view of such a machine is presented in figure 2.

Before a machine is started it needs to be configured. The configuration of a machine includes:

- the specification of machine inputs,
- adaptive process parameters,
- configuration of submachines.

Each machine may use any number of inputs. The inputs are connected to compatible outputs of other machines, to get some information from them.

The adaptive process of the machine may access the information from the inputs and may be controlled by a number of parameters. When finished it may present its gains to other machines by means of outputs (and results repository, but this kind of sharing results with other machines is out of the scope of this article, for more see [3], [4]).

The modular structure of our general data mining system is conducive to splitting more complex learning machines to a number of simpler, more specialized machines. Each machine is allowed to use other machines (called its submachines) to perform a part of its task. Thus the configuration of the submachines must be seen as a part of the configuration of the parent machine.

A. Information flow

Due to our view of learning machines and the input–output interconnections, any data analysis project may be represented as a directed acyclic graph with machines as vertices and input–output connections as edges.

A real life example of such a project is presented in figure 3. The project contains two machines for data loading (or data generation), one for training a classification machine composed of data standardization and Support Vector Machine (SVM) classifier and one to perform classification test of the model on data unseen during training. Following the arrows, we can observe the information flow between the machines:

- the training dataset becomes the input for the data standardization machine,
- the standardization machine collects statistics from the training data (accessed through the input) and exhibits

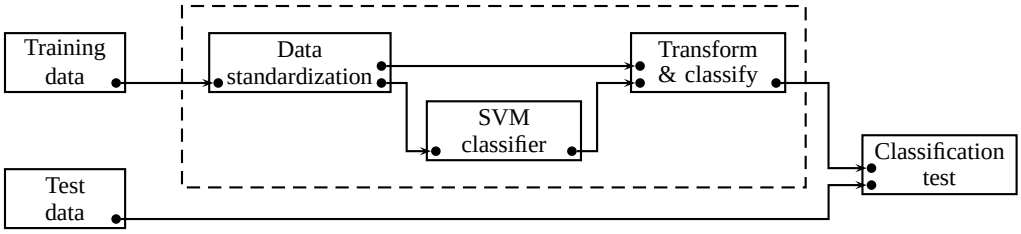

Fig. 3. An example of a DM project.

two outputs: the standardization routine (for standardization of other datasets according to the statistics obtained for the training dataset) and the result of the routine applied to the input dataset; the routine goes to the „Transform & classify" machine and the transformed dataset to the SVM classifier,

- the "Transform & classify" machine uses the standardization routine and the classifier to facilitate classification of the test data; its output is a classifier, which first transforms given data and then forwards the decisions of the SVM classifier.

The dashed rectangle encompasses three machines which may be treated as a single complex machine with a single input (the training data) and single output (the classification routine). We call such machine compositions *machine schemes* and treat them the same way as simple machines. The inner machines perform appropriate parts of the overall task, and in fact are submachines of the complex machine.

B. Machine complexity

Efficient meta-learning algorithms must be able to create, run and analyze different complex machine structures, but the machines should be examined in appropriate order: the simplest machines first, because usually it does not make sense to spend time on analysis of complex machines if simple ones perfectly solve the task. Therefore, in our system, we have provided tools for machine complexity measurement, as described in the further part of the article. Here we only need to mention, that machine complexity can not be calculated simply from the machine structure, but it must reflect also the time necessary to complete the learning processes for given input data. Sometimes more complex machine structures may run shorter, than much simpler ones, for example if we run the k Nearest Neighbors (kNN) algorithm on data described by 10000 features, it will take much more time, than running a sequence of two machines: first selecting randomly 10 features and the second being the same kNN machine as before, but run on the transformed data with just 10 features.

To try different machine configurations in the order of increasing complexity, the complexity must be estimated before the machine is run. The estimation may be accurate in some cases and very rough in others. To make it as accurate as possible, some information about the inputs must be provided. In the case of complex machine schemes a simulation of information flow must be performed to generate descriptions of outputs passed as inputs to other machines. The descriptions are collected and passed to the routines calculating the computation time requirements.

Our complexity estimation module takes into account all the components of the full learning machine configuration, i.e.:

- the inputs,
- adaptive process parameters,
- submachines configuration.

The module can automatically analyze machine structures of any depth, provided routines for simple machines description (generating descriptions of machine outputs and calculating the complexity estimates, given proper descriptions of the inputs). The descriptions have the form of dictionaries, so the representation is uniform and facilitates information exchange between different machines.

C. Meta-schemes

One of the most important areas of application of human experts knowledge is selection of the most reasonable learning machines combinations for particular problem. The knowledge about requirements of different machines and their eligibility to interact with others is used to filter out the promising machine structures to be tested: the most promising at the very beginning and the less promising later on.

As a counterpart of this experts knowledge in automated meta-learning, we have introduced *meta-schemes* which serve as templates of machine structures. They are schemes, which are not completely determined. Instead, they contain some placeholder(s) for different particular machines which are replaced by particular machines during the meta-search. The inverse replacements can easily generate meta-schemes from precisely defined schemes. For example, when we replace the "Data standardization" box of the scheme presented in figure 3 by a placeholder for a data transformation, we obtain a meta-scheme, that can be used to search for a transformation, that prepares the best form of the data for the classifier. In such place we may put any simple data transformation and also any complex structure of machines, which finally gives a dataset eligible for the input of the classifier. If we replace also the "SVM classifier" by a placeholder, we will get a meta-scheme with two placeholders (exactly as discussed later on and presented in figure 4), which may be a base for more sophisticated search for a

combination of machines (data transformers and classifiers) that can successfully act together.

IV. Complexity controlled meta-learning process

The space of potential solutions is usually very huge, but it does not mean that experts should be more effective than dedicated meta-learning algorithms which search through the model space in intelligent ways. From the other hand, even advanced experts have limited possibilities—it can be seen for instance from the difference of quality of solutions presented by experts in several competitions around computational intelligence.

The algorithm, presented below, can find solutions to different kinds of computational intelligence problems like classification, approximation, prediction, etc. Also, it may optimize different criteria, the selection of which, usually depends on the task which is to be solved. The solutions generated by our algorithm may be of simple or complex structure. They are searched for in a uniform process controlled with real complexity of algorithms (learning machines). Note that a single machine is not always of smaller complexity than another one of more complex structure but composed of submachines of small complexity. The complexity based control of meta-learning processes is of highest importance, because it helps avoid some traps which could crush the whole learning process.

Given a dataset representing the problem and a goal criterion, some learning machines can find a solution (with different efficiency and accuracy) but for some others the problem may be unsolvable (for example, may encounter convergence troubles because of their stochastic behavior, typical for some neural networks). Moreover, because of insolvability of the halting problem, we can not foresee if the learning processes will finish. The meta-learning algorithm, we propose, deals successfully also with such cases.

Our solution to these problems was inspired by the definition of complexity by Levin [12], [13]:

$$C_L(P) = \min_p \{c_L(p) : p \text{ is a program which solves } P\}, \tag{1}$$

where P is the problem to be solved and

$$c_L(p) = l(p) + \log(t(p)), \tag{2}$$

$l(p)$ is the length of program p and $t(p)$ is the time in which p solves P.

In more advanced meta-learning the Eq. 2 may be substituted by

$$c_{NiK}(p) = l(p) + \log(t(p)) - q(p), \tag{3}$$

where $q(p)$ is a function term responsible to reflect the inverse of an estimate of reliability of p, and p denotes a learning machine (the same applies to Eq. 1 when it is adapted to computational intelligence problems).

A. Complexity computation of learning machines

The meta-learning algorithm described below makes use of the complexity of learning machines browsed in the learning phase. In contrary to the Levin definition, our meta-learning is not able to explore infinite number of learning machines. However the spaces of candidate learning machines for meta-learning test tasks, may be infinite.

To compute the complexity of given learning machine it is necessary to have the following information about the configuration of such machine:

- meta-descriptions of all the machine inputs,
- configuration parameters of the machine,
- configuration of submachines (in the case of complex machines or schemes).

The *meta-descriptions* must exhibit all necessary information about inputs to facilitate accurate complexity computation for given machine. For example meta-description of a data table (a dataset in the form of *table*) input contains, between others, information about the number of instances, the number of attributes, the number of missing values and the numbers of ordered and unordered attributes. For some input types it may be necessary to have a functional form of a part of the input meta-description, it is needed for example for such inputs like classifiers or data transformers.

Additionally, the meta-learning algorithm needs a *complexity evaluator* for each type of learning machine. For example each classifier like kNN or SVM, each data transformer etc. needs its own complexity evaluator. It is necessary, because each learning machine has its own specific behavior. That behavior must be well known to the complexity evaluator to reliably compute the time and memory consumption basing on the configuration description (before the machine is created).

In the case of complex machines, i.e. when a given machine creates and uses some submachines, the machine complexity evaluator needs to call the evaluators of complexity of submachine(-s) and return the sum of all the complexities (independently for time and memory, of course). The submachines complexity evaluators are called with the information about meta-descriptions of the submachine inputs (for each submachine input), the adaptive process parameters and, if necessary, proper subconfigurations (configurations of submachines of submachines). This is the recurrent nature of complexity evaluation, which *de facto* reflects the recurrent nature of machine configuration and machines in run. Indeed, the complexity evaluators additionally have to produce meta-descriptions of their outputs, which may be essential to ensure accuracy of another machines complexity evaluators, for example in the case when a parent machine propagates an output of one child machine to an input of another child machine.

The computation of complexity of a scheme is equivalent to calculating the sum of complexities of the submachines computed by appropriate complexity evaluators in the order determined with the topological sort of input-output connections.

It may happen that for some learning machines it is impossible to determine their complexity because of their stochastic behavior. In such cases the approximation of complexity may be obtained. For example, by learning an approximation task for especially prepared dataset. The dataset may be created for an individual learning machine and single instance is created for information on single benchmark dataset (benchmark datasets are typical datasets for given tasks, for example typical classification or approximation benchmarks form UCI machine learning repository [14] may be used). Single instance, on the input part, consists of the characteristics from meta-descriptions of inputs of given machine, together with the configuration parameters and, on the output part, really consumed memory and time (by the learning process) for given benchmark dataset. In the learning process we obtain the approximator of complexity evaluator for given learning machine basing on given set of benchmark datasets.

B. Meta-learning algorithm

The main idea of the algorithm is to iterate in the main loop through the *programs* (*algorithms*, *learning machines*), constructed by a system of *machine generators* (described a little below), in the order of their complexity measured with Eq. 3 or in a simplified version with Eq. 2. In fact, the complexity which is used by the meta-learning algorithm to order machines, is a sum of two complexities: the first for the *learning part* and the second for the *test part*.

In general, our meta-learning algorithm may be seen as a loop of test estimations trials with a complexity control mechanism. Each generated machine is nested in the test procedure (adequate for the problem type and configured goal), then the test procedure starts and the loop supervises whether the complexity of the task does not exceed current complexity threshold. Such scheme of meta-learning fulfill the general meta-learning scheme presented in section II and figure 1.

The **goal of given meta-learning** algorithm is defined by:

- definition of the *stop condition*,
- definition of the *test* performed for machines generated by *machine generators*; the test is used to estimate usefulness of given machine,
- initialization of machine generators (via initial sets of appropriate machines).

The configurability of the meta-learning algorithm makes it universal, applicable to different types of problems and different goals.

a) The stop condition of the loop: As long as machines are generated by machine generators, the main loop may continue the job. However the process may be stopped for example when the goal is obtained (remember, that the goal may depend on the problem type and on our preferences). We may wish to:

- find the *best model* for given dataset in given amount of time,
- find the *best model* satisfying a goal condition with given threshold θ,
- find the *best model* satisfying a goal condition with given threshold θ, with as simple structure as possible,
- find a few *best models* which can be used as complementary and which satisfy a goal condition with given threshold θ,
- stop when the progress of objective function (test criterion) is smaller than a given ϵ.

Also, the term of the *best model* (or rather of *better model*) may be defined in different ways (on the basis of several concepts), however it is the simpler part of the algorithm. It is important to see that stopping criterion is not a problem —we just need to declare our preferences.

b) Start some test tasks: This step of the general meta-learning algorithm is devoted to defining and starting new test tasks. The algorithm keeps the started tasks in a special queue Q of specified limited size. A new task can be added only if the count of tasks in Q is smaller than the limit. The tasks in Q may run in parallel.

The tasks are constructed on the basis of machine configurations obtained from the set of generators. The procedure always gets the machine of the smallest complexity according to Eq. 3 or 2, considering all active generators (a meta-learner may change the set of machine generators up to its needs). The selected machine or rather its configuration is nested in a task which performs a test of the machine, for example in a cross-validation test. The type of the test and its parameters are also a subject of configuration. If the complexity of selected machine is not larger than the *current complexity* level, the current complexity is set to the maximum of current complexity and complexity of selected machine[1].

The outline of the procedure starting new tasks looks like:

```
1 procedure start tasks if possible;
2   while( started tasks count < limit )
3   {
4     m := find machine of simplest complexity in generators set
5     form new test task t for machine m
6     add t to Q
7     current_complexity :=
8             max(current_complexity, complexity_of(m))
9   }
```

c) Machine generators: The crucial role in the above symbolic code, plays the set of machine generators which is a source of machine configurations. Different machine generators may form significantly different solution spaces. Machine generators are also strongly goal–dependent (depend on the problem type and the criterion used for testing). The machine generators are asked to present or give single machines of the smallest complexity, one by one. The meta-learning procedure selects a machine of smallest complexity among the results obtained from all the generators. All of these ideas are realized very efficiently using appropriate data structures.

[1]It can not be simply set to the complexity of selected machine because it may happen (from different reasons) that a generator generates a new machine of smaller complexity.

The goal of using a set of generators instead of a single generator was that it is simpler to define several *dedicated* generators which are coherent, than a single universal one for any type of tasks. The generators may form different levels of abstractions in machines construction. They may be more or less sophisticated and produce more or less complex machines. The meta-learner may exchange results of the explorations between generators, integrating the possibilities of generators. The generators may be added or removed, during meta-learning, according to the needs of the meta-learning procedure. They may also adjust their behavior to the knowledge collected while learning, to produce new machines, more adequate to the experience, providing lower $q(p)$ of Eq. 3.

d) Complexity control of running tasks: In the step *wait for any task* algorithm waits for a naturally finished task or for a task which may consume more time or memory than it was assumed basing on the complexity of given task. All tasks are supervised, because otherwise, some of them could never finish or use too much time or too much system resources. When the consumed complexity of a task exceeds the threshold calculated for given task, the task is stopped and removed from the task queue Q. The estimated complexity of such task is increased with a fixed factor or according to the estimated progress of the task and the task is moved to the *quarantine*. If possible, (it depends on implementation of given machine) the task state is saved (via cache) to be restarted from the stopping-point, when the penalized complexity will become attractive again. Thus, the quarantine plays the role of a machine generator, which keeps the stopped tasks, for future use.

Similarly to the idea of machine examination in the order of increasing complexity, braking too complex processes resembles what human experts do when searching for attractive models, but here, instead of the fuzzy criterion of expert's patience we have a formal complexity-based test.

e) Results evaluation: Each finished task is removed from the task queue Q and the estimated quality of the tested machine, together with machine configuration and the results of learning, is moved to *results repository*. Partial results (current ranking of models) are available in real time (e.g. accessible from GUI).

All finished tasks help find more and more interesting solutions. Even if they do not provide very attractive solutions, they are a source of some meta-knowledge, helpful in further exploration, for example in estimation of the reliability of machines created by active generators for next generations. This information is very useful for adjustment of $q(p)$ from Eq. 3, which has crucial influence on the ordering of generated machines. For instance, if it is found, that a combination of given feature selection method works well with some classifier, we may promote such submachine structures in new machines.

C. Examples of machine generators

The simplest form of a machine generator is the one providing learning machines configuration from a predefined set. Such a generator must be capable of pointing to the simplest machine in the set. The same generator is used by our meta-learning algorithm to realize the *quarantine* for too complex machines.

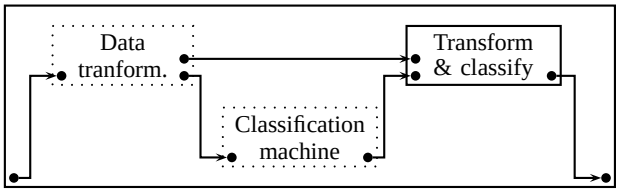

Fig. 4. A meta-scheme of data transformation and classification.

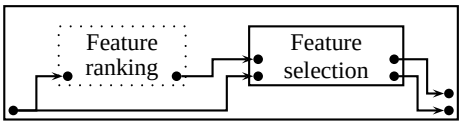

Fig. 5. A meta-scheme of feature selection transformation with placeholder for ranking machine.

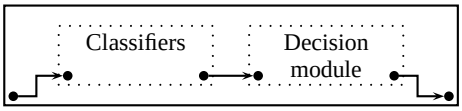

Fig. 6. A meta-scheme of a committee machine with placeholder for a number of classifiers and decision module.

The generators are free in the choice of knowledge used to generate machines. The *scheme based generator (SBG)* was designed to produce new machines using *meta-schemes*. A meta-scheme is a template which defines how to build structures of machines. Some examples of meta-schemes are presented in figures 4, 5 and 6.

Meta-schemes may contain machines, placeholders for machines and connections between machines inputs and outputs. SBGs fill the meta-schemes with particular machines obtaining complex machines.

The fact that the structure is more complex, does not imply a higher complexity of such new machine. Imagine a machine composed of a feature selection and a classifier (by filling the meta-scheme of figure 4). It may happen that the complexity of the feature selection is small and the transformation leaves small amount of features in the output dataset. The classifier trained on transformed data may have much smaller complexity, because of the dimensionality reduction, and final complexity of such composite model may be significantly lower than the complexity of the same classifier, when not preceded by the feature selection machine. This is a very important feature of our algorithm, because it facilitates finding solutions, even when the base algorithms are too complex, if only some compound machines can solve the problem effectively.

The meta-scheme of figure 4 enables creating machines which consist of any dataset transformation method and any classification machine. The choice of data transformation depends on initial configuration but also on newly produced

machines. Note that such compound, as a product of the meta-scheme, forms another classifier and it may be nested in another scheme. Also the transformation placeholder of this scheme may be filled directly by a data transformer or by an instance of a scheme which plays the role of dataset transformer (for example an instantiation of the meta-scheme presented in figure 5). The SBG type of generators should defend from producing tautology or nonsense (from computational intelligence point of view), however in general it is impossible to defend against every unnecessary or useless (sub-)solution.

Figure 5 presents a meta-scheme dedicated to feature selection. The role of the ranking machine (the placeholder) is to determine the importance order of features and the feature selection machine performs the selection of the top ranked features. A filled instance of that scheme may be nested in the previous meta-scheme to compose a classifier preceded by the feature selection.

Each machine generator may have its own tactic for building/composing new machines. In particular, the generator which composes machines from meta-schemes can be realized in a number of ways.

Figure 6 presents a general meta-scheme of a committee model. The classifiers can be inserted in the classifiers placeholder and a decision module in the other placeholder (it may be a voting/weighting/WTA or any other kind of decision module).

Another very important machine generator may be seen as a sub-meta-learning and is devoted to search for optimal (or close to optimal) configuration parameters for a given machine (including complex structures of machines). This machine generator produces a specialized test machine (*meta parameter search machine*) to search for *meta parameters*. By meta parameters of given machine we mean its configuration parameters which are declared to be searched automatically. Such parameters can be described by their types, interval of acceptable values, default values, interval of recommended values, recommended search strategy, etc. A meta parameter search machine tests given machine using one of several search strategies. The strategy should reflect behavior of the meta-parameter (linear, logarithmic, exponential or nominal). Several types of search are available for 1D and 2D depending on needs. The description of meta-parameters and their search methods provides a very interesting knowledge for the parameters search automation. The knowledge may be used by a machine generator to produce a series of independent machines and efficiently explore the space of possible machine configurations.

D. How it all works together

Meta-learning based on machine generators is a search process similar to what human experts do when analyzing data. The machine generators constructing machines according to figures 4–6 build machines, which are validated in proper order. The simplest machines are constructed by some substitutions to the meta-scheme of figure 4. One of the simplest transformations is data standardization, another one removes useless features with the filter of invariance[2]. They fit the first placeholder in the meta-scheme. Replacing the second box by a Naive Bayesian Classifier (NBC)[3] results in the instance of the meta-scheme of one of the smallest possible complexity. Thus, NBC trained on simply filtered data is one of the first candidate validated.

Not all the instances of this meta-scheme are so simple. We can also use Principal Components Analysis (PCA) as data transformation and a version of kNN with automated adjustment of k, obtaining quite computationally complex instance of the meta-scheme. Because of its large complexity, such machine is not tested at the very beginning of the search. It may get into the queue, even behind some models of more complex structure (for example composed of a data normalization, a simple feature selector and a classifier), but with more attractive time complexity prediction.

The complexity control also facilitates withdrawal of some methods, when their adaptive processes take too much time. It is quite natural, that for example a Support Vector Machine (SVM) training may be very difficult, when run on raw data, but after some feature selection, or other data transformation, the optimization process is very fast. In such cases the SVM which has been running for some time without success, is withdrawn, and other machines are tried. Otherwise, problematic machines could block the whole meta-learning.

The recursive nature of the meta-scheme presented in figure 6 facilitates taking advantage of what has been learned in the earlier stages of the search—the most successful (and most different) methods may be easily put into a committee to obtain even better or more stable results. It is not necessary to learn everything from scratch, when we start searching for committees, it is enough to combine the decisions of already created models, which may save a lot of time. It is also worth to notice, that evolutionary algorithms may be very easily implemented within our framework—it is enough to implement a machine generator capable of producing next generations and define the fitness function which will serve as the meta-learning validation criterion.

Small number of simple machine generators allows us to create quite complex machines and search for optimal configuration of their components. Experts meta-knowledge used to define an adequate set of meta-schemes and the mechanism of complexity control significantly reduce the search space, while not resigning from the most attractive solutions.

Obviously, providing unreasonable machine generators (for example generating very large number of similar machines of simple structure but poorly performing) or misleading complexity estimators, may easily spoil the whole meta-learning process, so all the components of the algorithm must be carefully selected.

[2]It removes each feature, which variance is equal to zero.

[3]Our implementation of NBC works with both nominal and continuous features.

V. SUMMARY

The meta-learning algorithm we propose is based on machine generators and complexity control. Meta-schemes restrict testing to only such machine architectures, that we regard as sensible. We provide mechanisms for estimation of machine (and model) complexity, before starting adaptive processes and use the estimates to test machines in proper order and to control the search process. Validating candidate machines in the order of increasing complexity guarantees success in the pursuit for suboptimal models—if there is an accurate structure (compatible with the meta-schemes), then it will be found in a finite time (the smaller complexity, the earlier) for the same reasons for which breadth first search successfully explores possibly infinite trees.

Our system supplies tools for easy meta-level activity, so that meta-knowledge may be easily extracted from data mining projects. Our algorithm collects such information to improve further search stages, for more efficient selection of committee members etc. More advanced methods for collecting, exchange and exploiting meta-knowledge will be one of our most important interests in the future.

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

REFERENCES

[1] K. Grąbczewski and N. Jankowski, "Meta-learning architecture for knowledge representation and management in computational intelligence," *International Journal of Information Technology and Intelligent Computing*, p. 27, 2007, (in print).

[2] ——, "Toward versatile and efficient meta-learning: Knowledge representation and management in computational intelligence," in *IEEE Symposium Series on Computational Intelligence*. USA: IEEE Press, 2007, pp. 51–58.

[3] N. Jankowski and K. Grąbczewski, "Learning machines information distribution system with example applications," in *Computer Recognition systems 2*, ser. Adavances in Soft Computing. Springer, 2007, pp. 205–215.

[4] ——, "Gained knowledge exchange and analysis for meta-learning," in *Proceedings of International Conference on Machine Learning and Cybernetics*. Hong Kong, China: IEEE Press, 2007, pp. 795–802.

[5] 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.

[6] 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.

[7] 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

[8] 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.

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

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

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

[12] L. A. Levin, "Universal sequential search problems," *Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, vol. 9, 1973.

[13] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, ser. Text and Monographs in Computer Science. Springer-Verlag, 1993.

[14] C. J. Merz and P. M. Murphy, "UCI repository of machine learning databases," 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.