Meta-learning with machine generators and complexity controlled exploration

Krzysztof Grąbczewski and Norbert Jankowski

Department of Informatics
Nicolaus Copernicus University
Toruń, Poland
`http://www.is.umk.pl/`
`{norbert|kgrabcze}@is.umk.pl`

Abstract. We present a novel approach to meta-learning, which is not just a ranking of methods, not just a strategy for building model committees, but an algorithm performing a search similar to what human experts do when analyzing data, solving full scope of data mining problems. The search through the space of possible solutions is driven by special mechanisms of machine generators based on meta-schemes. The approach facilitates using human experts knowledge to restrict the search space and gaining meta-knowledge in an automated manner. The conclusions help in further search and may also be passed to other meta-learners. All the functionality is included in our new general architecture for data mining, especially eligible for meta-learning tasks.

1 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.

A *learning problem* can be defined as $\mathcal{P} = \langle D, \mathcal{M} \rangle$, where $D \subseteq \mathcal{D}$ is a *learning dataset* and $\mathcal{M}$ is a *model space*.

In computational intelligence attractive models $m \in \mathcal{M}$ are determined with learning processs:

$$L^p : \mathcal{D} \to \mathcal{M}, \tag{1}$$

where p defines the parameters of the learning machine. This view of learning encircles many different approaches of supervised and unsupervised learning including classification, approximation, clustering, finding associations etc. Such definition does not limit the concept of search to specific kinds of learning methods like neural networks or statistical algorithms, however such reduction of model space is possible in practice.

In real life problems, sensible solutions $m \in \mathcal{M}$ are usually so complex, that it is very advantageous to decompose the given problem $\mathcal{P} = \langle D, \mathcal{M} \rangle$ into subproblems:

$$[\mathcal{P}_1, \ldots, \mathcal{P}_n] \tag{2}$$

where $\mathcal{P}_i = \langle D_i, \mathcal{M}_i \rangle$. In this way, the vector of solutions of the problems $\mathcal{P}_i$ constitutes a model for the main problem $\mathcal{P}$:

$$m = [m_1, \ldots, m_n], \tag{3}$$

and the model space gets the form

$$\mathcal{M} = \mathcal{M}_1 \times \ldots \times \mathcal{M}_n. \tag{4}$$

In practice $\mathcal{M}$ is usually a subspace of $\mathcal{M}_1 \times \cdots \times \mathcal{M}_n$, because the submachines may extract not only the component needed for the main model but also some other information. It is not a problem, because in such cases, the final model is just a simple projection of the whole complex model.

The solution constructed by a decomposition is often much easier to find, because the main task gets reduced to a series of simpler tasks: model m_i solving the subproblem $\mathcal{P}_i$, is the result of a learning process

$$L_i^{p_i} : \mathcal{D}_i \rightarrow \mathcal{M}_i, \quad i = 1, \ldots, n, \tag{5}$$

where

$$\mathcal{D}_i = \prod_{k \in K_i} \mathcal{M}_k, \tag{6}$$

and $K_i \subseteq \{0, 1, \ldots, i-1\}$, $\mathcal{M}_0 = \mathcal{D}$. It means that the learning machine $L_i^{p_i}$ may take advantage of some of the models $m_1, \ldots, m_{i-1}$ learned by preceding subprocesses and of the original dataset D of the main problem $\mathcal{P}$. Naturally, also parameters $p = (p_1, \ldots, p_n)$.

So, the main learning process L^p is decomposed to the vector

$$[L_i^{p_i}, \ldots, L_n^{p_n}]. \tag{7}$$

Such decomposition is often very natural: a standardization or feature selection naturally precedes classification, a set of classifiers precedes a committee module etc. Note that, the subprocesses need not to be dependent on all preceding subprocesses, so such decomposition has natural consequences in the possibility of parallelization of problem solving.

A real life example of a learning process decomposition is presented in figure 1, where a classification committee is constructed, but member classifiers need the data transformed before they can be applied. The structure of the project directly corresponds to both the learning process and the final model decomposition. The rectangle including all small boxes but “Data”, depicts the whole learning process, which, given dataset, is expected to provide a classification routine. For different kinds of analysis like testing classification accuracy etc. it must be treated as a whole, but from the formal point of view each inner rounded rectangle is a separate process solving its task and providing its model.

Because each DM process is a directed acyclic graph, it is easy to show the composite process and composite model it corresponds to. The model of figure 1 may be

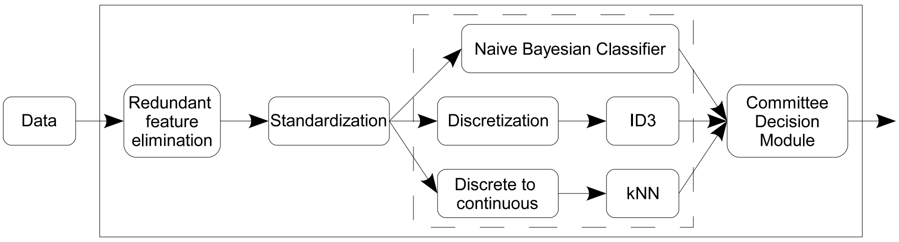
Data
Redundant feature elimination
Standardization
Naive Bayesian Classifier
Discretization
ID3
Discrete to continuous
kNN
Committee Decision Module

Fig. 1. An example of a DM project.

decomposed as

$$\begin{aligned}
m_{rfe} &= L_{rfe}(Data) \\
m_{std} &= L_{std}(m_{rfe}) \\
m_{nbc} &= L_{nbc}(m_{std}) \\
m_{discr} &= L_{discr}(m_{std}) \\
m_{id3} &= L_{id3}(m_{discr}) \\
m_{dtc} &= L_{dtc}(m_{std}) \\
m_{knn} &= L_{knn}(m_{dtc}) \\
m_{comm} &= L_{comm}(m_{nbc}, m_{id3}, m_{knn})
\end{aligned} \tag{8}$$

The subscripts are easy to decode, when compared to the figure 1. Each of the components learns some part of the final model, which has a corresponding structure.

Such general and uniform foundations of our DM system facilitate solving problem of any kind, requiring any structural complexity, provided appropriate components. It is especially important when undertaking meta-learning challenges, where we must try many different methods, from simple ones to those of large complexity. Nontriviality of model selection is evident when browsing the results of NIPS 2003 Challenge in Feature Selection [1, 2] or WCCI Performance Prediction Challenge [3] in 2006.

Some meta-learning approaches [4–7] are based on data characterization techniques (characteristics of data like the numbers 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.

This article presents some details of the structure and functionality of a meta-learning machine, naturally implemented within the architecture of our recently created system. It is an efficient algorithm, which can find many interesting solutions and is a good starting point to even better algorithms, 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, capable of gathering and exploiting meta-knowledge.

2 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 side, even the most advanced expert is limited in some ways—it can be seen for instance when browsing the difference of quality of solutions, presented by experts in competitions in the area of 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 complexities. 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, in accordance with insolvability of the halting problem, some learning processes may be infinite. 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 [8, 9]:

$$C_L(P) = \min_p \{c_L(p) : p \text{ is a program which solves } P\}, \tag{9}$$

where P is the problem to be solved and

$$c_L(p) = l(p) + \log(t(p)), \tag{10}$$

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

In more advanced meta-learning the Eq. 10 may be substituted by

$$c_{NiK}(p) = [l(p) + \log(t(p))] \cdot q(p), \tag{11}$$

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 (or a submachine) (the same applies to Eq. 9 when it is adapted to computational intelligence problems).

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 below), in the order of their complexity measured with Eq. 11 or, in a simplified version, with Eq. 10. 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 estimation 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.

An outline of the meta-learning algorithm may be presented as:

```
1 procedure ML;
2 while( stop_condition != true )
3 {
4   start tasks if possible
5   check the complexity of running tasks
6   analyze finished tasks
7 }
```

The following subsections describe details of subsequent parts of this algorithm.

2.1 The stopping 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* in given time for given dataset,
- find the *best model* satisfying a goal condition with given threshold θ,
- find the *best model* satisfying a goal condition with given threshold θ, and of as simple structure as possible,
- find several *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 *best model* (or rather of *better model*) may be defined differently (based on several concepts), however it is the simpler part of the algorithm. It is important to see that stopping criterion is not a problem itself—we just need to declare our preferences.

2.2 Starting new tasks

Line number 4 of the procedure ML is devoted to starting new tasks. The algorithm keeps the started tasks in a special queue Q of a specified, limited size. A new task can be added only if the number 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 a set of generators. The procedure always gets the machine of the smallest estimated complexity, according to Eq. 11 or 10, 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 is also a subject to configuration. If the complexity of selected machine was not bigger than *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:

```
procedure start tasks if possible;
 while ( started tasks count < limit )
 {
  m := find machine of simplest complexity in generators set
  form new test task t for machine m
  add t to Q
  current_complexity := $\max($current_complexity, complexity_of(m)$)$
 }
```

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.

To enable the calculation of *machine complexity*, for each machine, there is a function estimating the complexity on the basis of the machine configuration and the inputs it gets. In the case, when the estimation is not possible, the complexity is approximated from averaged past observation of the behavior of the method on different inputs. In both cases the estimated complexity is additionally weighted to obtain similar runtime behavior of machines which declare the same (or very similar) complexity.

The goal of using a set of generators instead of a single generator was that it is simpler to define several *dedicated* generators, 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 ML 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. 11.

[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.

2.3 Complexity control of running tasks

The tasks which are running, must be checked whether they don't consume more time or memory than planned. All tasks are supervised, because otherwise, some of them could use too much time and block the resources for other tasks. When the (realized/-consumed) complexity of a task exceeds the threshold calculated on the basis of the value of *current_complexity*, the task is stopped and removed from the tasks 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 (outside of memory) to be restarted from the stopping-point, when *current_complexity* gets adequately large. Thus, the quarantine plays the role of a machine generator, which stores 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.

2.4 Analysis of finished tasks

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. 11, 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.

2.5 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.

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 2, 3 and 4. Meta-schemes may contain machines, placeholders for machines and connections between machines inputs and outputs. SBGs fill the meta-schemes with particular machines provided in some sets obtaining complex machines. The fact that the structure is more

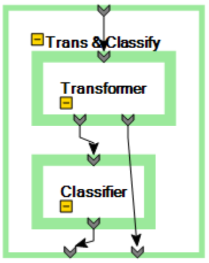

Fig. 2. A meta-scheme of data transformation and classification.

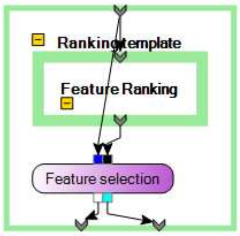

Fig. 3. A meta-scheme of feature selection transformation with placeholder for ranking machine.

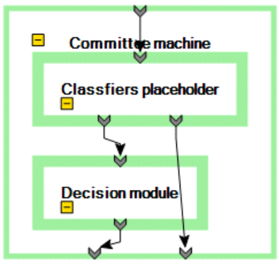

Fig. 4. A meta-scheme of a committee machine with placeholder for a number of classifiers and decision module.

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 2). 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 2 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 the meta-scheme of figure 3). The SBG type of generators should avoid producing tautology or nonsense (from computational intelligence point of view), however, in general, it is impossible to prevent the generators from providing unnecessary or useless (sub-)solutions.

Figure 3 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. Especially the generator which composes machines from meta-schemes can be realized in a number of ways.

Figure 4 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). We have implemented several search strategies for 1D and 2D. The description of meta-parameters and their search methods provide 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.

2.6 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 construct machines according to figures 2–4. The generated machines are validated in proper order. The simplest machines are constructed by some substitutions to the meta-scheme of figure 2. 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

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

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

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 4 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.

3 Summary

We have presented a meta-learning algorithm based on machine generators and complexity control. Using meta-schemes restricts testing to only such machine architectures, that we regard as sensible. Our definition of machine (and model) complexity, and the mechanisms for their estimation (before starting adaptive processes) and control (during their runs), facilitate testing machines in proper order. 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, for the same reasons, for which the 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 was supported by the Polish Ministry of Science with a grant for years 2005–2007.

References

1. Guyon, I.: Nips 2003 workshop on feature extraction. http://www.clopinet.com/-isabelle/Projects/NIPS2003/ (December 2003)

2. Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.: Feature extraction, foundations and applications. Springer (2006)
3. Guyon, I.: Performance prediction challenge. http://www.modelselect.inf.ethz.ch/ (July 2006)
4. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann (June 2000) 743–750
5. Brazdil, P., Soares, C., da Costa, J.P.: Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Machine Learning **50**(3) (2003) 251–277
6. Bensusan, H., Giraud-Carrier, C., Kennedy, C.J.: A higher-order approach to meta-learning. In Cussens, J., Frisch, A., eds.: Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming. (2000) 33–42
7. Y.H., Peng, Falch, P., Soares, C., Brazdil, P.: Improved dataset characterisation for meta-learning. In: The 5th International Conference on Discovery Science, Luebeck, Germany, Springer-Verlag (January 2002) 141–152
8. Levin, L.A.: Universal sequential search problems. Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian)) **9** (1973)
9. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications. Text and Monographs in Computer Science. Springer-Verlag (1993)