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

Norbert Jankowski and Krzysztof Grąbczewski

Department of Informatics
Nicolaus Copernicus University
Meta-learning introduction I

Meta-learning is learning how to learn.

- To perform meta-level analysis of *learning from data* one needs a robust and flexible system for different kinds of learning with uniform management of learning machines and their results.
- In our approach the term *meta-learning* encompasses the whole complex process of model construction:
  - learning and adjustment parameters for any parts of the machine hierarchy,
  - construction of machines hierarchies (with data transformation methods and other adaptive processes)
  - performing model validation and complexity analysis, etc.
Our long-range goal is to eliminate human interactivity in the processes and obtain meta-learning algorithms which will outperform human-constructed models.

Humans: knowledge + experience + intuition.

Meta-learning: definable precise criteria.

Our approach
- Searching through large spaces of possible models
- The simplest machines should be tried first!
- Complexity for meta-learning!
Learning machine may get some data as input and as a result returns some output(s).

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

Modular structure—splitting of more complex learning machines to specialized submachines.
The input–output interconnections compose a directed acyclic graph (DAG) with machines as vertices and input–output connections as edges.

The dashed rectangle encompasses three machines — machine scheme — it may be treated in the same way as simple machines.
Meta-learning algorithm I

START
initialize

stop condition

evaluate results

start some test tasks

wait for any task

finalize

yes

STOP

no

wait for any task
The **goal definition** for meta-learning

- 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).

**Universal meta-learning algorithm**

Proper configuration makes the algorithm applicable to different types of problems
The stop condition directly corresponds to the particular goal:

- find the **best model** for given dataset in given amount of time,
- find the **best model** satisfying a goal condition with given threshold $\theta$,
- find the **best model** satisfying a goal condition with given threshold $\theta$, 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 $\theta$,
- stop when the progress of objective function (test criterion) is smaller than a given $\epsilon$. 
Efficient meta-learning must create, run and analyze different complex machine structures, but...

- The simplest machines first—machines analyzed in the order of increasing complexity.
- Need of tools for machine complexity measurement.
- Machine complexity should reflect complexity of structure and time.
Suitable for different computational intelligence problems like classification, approximation, prediction, etc. It may optimize different criteria—depending on the goal definition.

The complexity by Levin:

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

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:

\[
c_{NiK}(p) = l(p) + \log(t(p)) - q(p),
\]

where \( q(p) \) is a function term reflecting the inverse of an estimate of reliability of \( p \) (learning machine).
To compute the machine complexity the following information is necessary:

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

**Meta-descriptions**

All necessary information about inputs to facilitate accurate complexity computation for given machine. For example in case of data table: the number of instances, the number of attributes, the number of missing values and the numbers of ordered and unordered attributes. Sometimes functional form of meta-description.
A **complexity evaluator** for each type of machine.

Additionally evaluators have to produce meta-descriptions of their outputs.

For complex machines:

- Evaluators need to call the evaluators of complexity of submachine(-s).
- The submachines complexity evaluators are called with appropriate information.
- Complexity of a scheme = the sum of complexities of the submachines.
Machines either finish within the (time & memory) constraints or are stopped and moved to the quarantine.

The quarantined machines may be restarted (continued), when the penalized complexity becomes attractive again.

Braking too complex processes resembles what human experts do when searching for attractive models, but here it is based on a formal complexity-based test.
Meta-schemes are templates of machine structures (not completely determined schemes).

- May contain machines, placeholders for machines and connections between machines inputs and outputs.
- Example meta-scheme of data transformation and classification:

![Diagram of data transformation and classification meta-scheme]

- Data transform.
- Transform & classify
- Classification machine
Machine generators provide learning machine configurations for the meta-search loop.

- Different machine generators explore different solution spaces
- Using a set of generators instead of a single generator helps define dedicated, coherent generators.
- May form different levels of abstractions in machines construction.
- May be added or removed, during meta-learning.
- May adjust their behavior to the knowledge collected while learning.

The simplest machine generator is the one providing learning machines configurations from a predefined set.
The scheme based generator (SBG) was designed to produce new machines using meta-schemes. Examples of meta-schemes for SBG:

- Data transform.
- Classification machine
- Transform & classify

Example transformations: data standardization, PCA, removing useless features with the filter of invariance.

Example classifiers: Naive Bayesian Classifier (NBC), kNN, auto-kNN, decision trees, SVM.
The recursive nature of the "Committee" meta-scheme facilitates combining machines validated in the earlier stages to obtain even better or more stable results.

Small number of simple machine generators allows to create quite complex machines.

Experts meta-knowledge helps define an adequate set of meta-schemes.
The fact that the structure of a scheme is more complex, does not imply a higher complexity of such new machine. Example:

- Machine composed of a feature selection and a classifier.
- Feature selection machine of small complexity may leave small number of features in the output dataset.
- The classifier trained on transformed data may have much smaller complexity, because of the dimensionality reduction.
- The overall complexity of the scheme may be less than that of single classifier (not preceded by the feature selection).
Meta-parameter search

- Searching for quasi-optimal configuration parameters (for complex machines too)
- Meta-parameters:
  - Declaration of meta-parameter
  - Domain descriptions, recommended search strategy
How it all works together I

- Machine generators provide configurations of machines to be validated.
- Meta-learning process.
  - Resembles what human experts do when analyzing data.
  - Validates the machines suggested by the generators.
  - Controls machine complexity to stop long-lasting learning subprocesses, to maintain the proper order.
- Meta-schemes+Generators restrict testing to only such machine architectures, that we regard as sensible.
- Validating candidate machines in the order of increasing complexity guarantees success in the pursuit for suboptimal models.
- More advanced methods for collecting, exchange and exploiting meta-knowledge will be our most important interests in the future.
A class of efficient algorithms, which can find many interesting solutions.

Open gates to easy implementation of more advanced meta-learning techniques, gathering and exploiting meta-knowledge.