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

Krzysztof Grąbczewski and Norbert Jankowski

Department of Informatics
Nicolaus Copernicus University
Abstract

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 solutions for an advanced data mining system, efficient not only in model construction for given data, but specially in meta-learning.
Most of current Data Mining systems can not discover models in really automated and autonomous way. Advanced Data Mining systems for complex model construction and analysis must provide:

- A unified view of most aspects of handling CI models model construction, a general input–output representation for information exchange.
- Easy and uniform access to model configuration; possibility to describe the characteristic of params (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 with uniform access.
Why the new architecture for meta-learning is indispensable

- Tools for estimation of model *relevance* (according to the specific goal);
  analysis of reliability, complexity and statistical significance.
- Tools for 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.
- Versatile time and memory management to guarantee optimal usage of the resources;
  model cache systems and unification framework.
- Simple and highly versatile Software Development Kit (SDK) for programming system extensions.
Versatility

- In data mining system *versatility* means that many different kinds of data sets can be easily analyzed with many different kinds of tools. [“tabular” data, text data, bioinformatic sequences or microarrays, etc.]
- Needs of miscellaneous transformations before the final knowledge extraction.
- Knowledge extraction by: statistics, machine learning, neural networks, etc.
- Solving different optimization problems (classification, regression, clustering, etc.).
System components unification

- Needs of an abstract definition of a model.
- Project/Repositories/Model-Base/Config-Base offer all common functionality.
- Such unification is necessary not only as a programming technique.
- Unification facilitating information interchange.
- Unification for reduction of memory and computational time consumption (reusing models, cache).
- Exploration of the models space without deep knowledge about the particular elements of the system (crucial in meta-learning).
Common/General input–output scheme of model

- All components use the same language to describe inputs and outputs.
- It allows to combine the components simply by user interaction (GUI/SDK).
- Interacting models: adaptive models (simple/complex), advanced testing models, etc.
- A general data analysis framework must provide some standard methods for statistical significance testing.
Each component should present its most interesting gains to the system THEN interactive user or other models can take advantage of them.

It becomes especially important when meta-learning algorithms are constructed.

A uniform results repository makes analysis of model results model-independent and facilitates very deep meta-level analysis with simple means.

The results are available even when the model cleared.

The ideal of commentators.

Query/Series/Transformer
SDK

- The external developers are not 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.

User interface

- The ease of navigation within 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.
Methods and models

**Method**

Method (learning method) = adaptive algorithms

**Model**

Model = the final result of application of a method with some particular parameters to particular input data.

[Fully-functional model performing approximation, classification, tests, data loading or transformers, etc.]

Unified view of model without distinction of data preprocessing and proper model building. [Preprocessing mostly = misunderstanding. Transformations before final model! CV...]
Models abstraction

- Model configuration: parameters, submodels configurations, etc.
- Fitting different levels of abstraction.
- Data transformation sharing...
- Power savings in data transformations: we do not need to copy data, when we select a subset of features or vectors.
**Inputs vs parameters**

*Inputs*: to provide means for exploiting outputs of other models

*Parameters*: do not interfere with external models JUST specify how the adaptive process will operate on inputs to generate outputs and results.

**Outputs vs results**

Both are the effects of the adaptive process of the model.

*Results* are deposited in a special repository, which makes them available even after the model itself is released (static results).

*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).
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.
Information exchange and complex model structures

Modular structure

Any model may contain some submodels of any type and any level of abstraction.
Useful in: testing models (repeaters, monte carlo, cross-validation), ensembles, meta-level methods and others.

Input–output interfaces

Models may be connected using input and output interfaces.

Type of connection determine one deeper or shallower understanding by second model. Consider two output types: SVM-classifier and classifier.
A *scheme box* is a 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.

- A scheme box may have inputs and outputs.
- Scheme may also play the role of a submodel.
- 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).
- There are no limits on the types of submodels of scheme box.
- Scheme boxes may define templates by abstract boxes.
Configuration of a CV Repeater for classification.

- Dataset
- CV Distr Board
- CV Repeater
- kNN
- SVM
- TestA
- TestB
CV Repeater of figure at runtime.

- Dataset
- CV Distr Board
- CV Repeater
- R1/F1
  - Distr
  - kNN
  - SVM
  - TestA
  - TestB
- R1/F2
  - Distr
  - kNN
  - SVM
  - TestA
  - TestB
- R2/F1
  - Distr
  - kNN
  - SVM
  - TestA
  - TestB
- R2/F2
  - Distr
  - kNN
  - SVM
  - TestA
  - TestB

NiK/SSCIDM07 (NCU)  Meta-Learning Architecture  17 / 18
An example of meta-learning model with transformation template.